sciendo
Proceedings on Privacy Enhancing Technologies ; 2020 (4):220–238

Gunes Acar*, Steven Englehardt*, and Arvind Narayanan*

No boundaries: data exfiltration by third parties embedded on web pages

Abstract: We investigate data exfiltration by third-party scripts directly embedded on web pages. Specifically, we study three attacks: misuse of browsers' internal login managers, social data exfiltration, and whole-DOM exfiltration. Although the possibility of these attacks was well known, we provide the first empirical evidence based on measurements of 300,000 distinct web pages from 50,000 sites. We extend OpenWPM's instrumentation to detect and precisely attribute these attacks to specific third-party scripts. Our analysis reveals invasive practices such as inserting invisible login forms to trigger autofilling of the saved user credentials, and reading and exfiltrating social network data when the user logs in via Facebook login. Further, we uncovered password, credit card, and health data leaks to third parties due to wholesale collection of the DOM. We discuss the lessons learned from the responses to the initial disclosure of our findings and fixes that were deployed by the websites, browser vendors, third-party libraries and privacy protection tools.

Keywords: JavaScript, privacy, tracking

DOI 10.2478/popets-2020-0070
Received 2020-02-29; revised 2020-06-15; accepted 2020-06-16.

1 Introduction

The vast majority of websites today embed one or more third-party scripts in a first-party context, i.e. without any iframes. This practice is fundamentally insecure, because it negates the protections of the Same-Origin Policy and gives third-party scripts access to virtually all sensitive data on the page. Our goal in this paper is to discover if third parties are actually abusing these privileges to exfiltrate sensitive data.

Specifically, we examine three attacks. We chose these particular attacks because they had not been measured at scale before and have severe consequences for users including the leakage of their passwords, health data and credit card details. The attacks, and the measurement techniques we develop to examine them, are not exhaustive. Instead, they serve as three specific instances through which we examine the more systemic problem of loading untrusted JavaScript in a first-party origin.

1. **Login manager misuse.** Browsers login managers automatically fill in previously saved credentials for known form fields. Scripts may misuse this feature by inserting invisible login forms into pages to trigger browsers' login manager and read the inserted credentials. This attack was known to browser vendors at the time of the measurements, but only some browsers had mitigations for this attack and they were imperfect (Section 4).
2. **Social data exfiltration.** Many websites support authentication through Facebook Login[1] or similar federated identity providers. When users login through these providers, user data held by these providers may also be accessed and exfiltrated by third-party scripts. In some, but not all, cases this happens with the cooperation of the first party (Section 5).
3. **Whole-DOM exfiltration.** To analyze user interactions on websites, certain third-party scripts serialize the entire DOM (Document Object Model, the tree of objects that constitutes the web page) and send it to their servers. These are providers of "session replay" services and their wholesale collection of page data may cause leakage of sensitive data (Section 6).

We set out to detect these attacks on a large scale, which required a number of new measurement methods and improvements to existing methods (Section 3). We built our measurement framework by extending OpenWPM, an open-source web privacy measurement tool [2]. First, we added a number of missing features to

***Corresponding Author: Gunes Acar:** imec-COSIC KU Leuven, E-mail: gunes.acar@kuleuven.be
***Corresponding Author: Steven Englehardt:** Mozilla, E-mail: senglehardt@mozilla.com
***Corresponding Author: Arvind Narayanan:** Princeton University, E-mail: arvindn@cs.princeton.edu

OpenWPM, including the ability to precisely attribute behaviors such as reading from DOM to specific third-party scripts. Second, we created instrumentation code to detect each of the three attack categories of interest. This required combined analysis of data from several different instrumentation domains including JavaScript function calls, HTTP messages and DOM elements.

Using a full-fledged web browser automated and instrumented as described above, we ran a separate crawl of 300,000 web pages for each of the three attacks. Our findings show that tracking and analytics scripts engage in highly questionable practices such as injecting invisible login forms to trigger browser login manager and exfiltrate personal data (Section 4). Further, collection of entire DOM by session replay scripts cause unwitting exfiltration of personal and sensitive information such as credit card details, medical details and passwords (Section 6).

Listing 1 Snippet of the `behavioralengine.com` code that inserts an invisible form to trigger browser autofill and exfiltrates the MD5 hash of the user's email address.

```
checkId: function(self) {
  var container = document.createElement('div
      ');
  container.id = 'be-container';
  container.style.display = 'none';
  var form = document.createElement('form');
  form.attributes.autocomplete = 'on';
  var emailInput = document.createElement('
      input');
  emailInput.attributes.vcard_name = '
      vCard.Email';
  emailInput.id = 'email';
  emailInput.type = 'email';
  emailInput.name = 'email';
  form.appendChild(emailInput);
  var passwordInput = document.createElement(
      'input');
  passwordInput.id = 'password';
  passwordInput.type = 'password';
  passwordInput.name = 'password';
  form.appendChild(passwordInput);
  container.appendChild(form);
  document.body.appendChild(container);
  window.setTimeout(function() {
    if (self.emailRegexp.test(
        emailInput.value))
      self.sendHash(self, MD5(
          emailInput.value));
    document.body.removeChild(container)
  }, 1e3)
```

While we have provided the first empirical evidence that these attacks are widespread, their *possibility* is well known [3], and is highlighted in most web security guides [4]. We argue that a combination of commercial and technical reasons are responsible for the limited deployment of defenses against these attacks.

We first publicly released the our findings in 2017-18 (with the appropriate disclosure to relevant parties, whenever possible). This time gap gives us a unique opportunity to review the effects of a large-scale analysis and disclosure of web privacy vulnerabilities. Our review shows that the specific vulnerabilities we reported have largely been addressed, but the root causes of the problem remain intact. We suggest two potential paths forward based on this analysis (Section 7).

2 Background and related work

Personally Identifiable Information (PII) leakage. The ability of third-party trackers to compile information about users is aided by PII leaks from first parties to third parties. Such leaks allow trackers to attach identities to pseudonymous browsing histories.

PII leakage can be intentional or inadvertent. First parties may send PII to third parties for business purposes. Alternately, the user's email address or user ID may be carelessly embedded in a URL (e.g., `example.com/profile?user=userID`) and thus leak to third-party servers via the `Referer` [sic] header. `Referer` header leaks are amplified by the availability of the leaking URI via the `document.referrer` API [5] on subsequent page load or in embedded iframes.

Until recently, PII leakage could only be prevented if a first party took care to ensure no user information was included in their site's URLs. However, the recent `Referrer-Policy` standard has made it possible for browser vendors, extension authors, and sites to control the scope of the `Referer` header sent to third-party servers [6]. The standard allows policies to be specified to strip `Referer` headers down to the URL's origin or remove it entirely for cross-origin resources.

Measurement studies. Early work by Krishnamurthy and Wills found that both intentional and unintentional PII leakage were common in social networks [7, 8]. In follow-up studies, Krishnamurthy et al. [9] and Mayer [10] showed that leakage is common across the web—around half of the websites studied leaked user identifiers to a third party. Mayer found PII leaks to as many as 31 separate third parties on a single site, and discovered an instance of dating profile intentionally leaking profile data to two major data brokers [10].

More recent studies include detection of PII leakage to third parties in mobile apps [11–13], in enterprise network traffic [14], PII leakage in contact forms [15], and data leakage due to browser extensions [16]. In our paper we discover PII exfiltration by tracking scripts that misuse login managers (Section 4), social APIs (Section 5) and session replay scripts (Section 6).

Trackers wishing to follow users across multiple devices have developed deterministic and probabilistic techniques collectively referred to as "cross-device tracking" [17]. PII exfiltration is a plausible route to deterministic cross-device tracking [17]; our work thus contributes to the understanding of cross-device tracking. IP address and browsing history are often used as probabilistic features [18].

Obfuscated PII. A common problem faced by authors of past work (and by us) is that PII may be obfuscated before collection. When the data collection is crowdsourced [11, 12] rather than automated, there is the further complication that the strings that constitute PII are not specified by the researcher and thus not known in advance. Early work recognizes the possibility of obfuscated PII, but accepts it as a limitation [7, 8, 10]. Various approaches are taken in the work that follows. Ren et al. employ heuristics for splitting fields in network traffic and detecting likely keys; they then apply machine learning to discriminate between PII and other fields [11]. Hedin et al. [19] and De Groef et al. [20] use information flow analysis to trace exfiltration of sensitive data. Starov et al. apply differential testing, that is, varying the PII entered into the system and detecting the resulting changes in information flows [15]. Brookman et al. [17], Starov et al. [16], Ren et al. [21] and Reyes et al. [13] test combinations of encodings and/or hashes, which is most similar to the approach we take in Section 3.4.

3 Methods

3.1 Measurement configuration

To study each attack, we crawled 300,000 pages from 50,000 websites. We sampled the 50,000 sites based on Alexa rank as follows: all of the top 15,000 sites; a random sample of 15,000 sites from the Alexa rank range [15,000 100,000); a random sample of 20,000 sites from the range [100,000, 1,000,000). This combination allowed us to observe the attacks on both high and low traffic sites. On each of these 50,000 sites we visited six pages: the front page and a set of five other pages randomly sampled from the internal links on the front page. We limited our sampling to the internal pages from the same domain. Pages that redirected to other domains during crawls were excluded from analyses. Since we ran a separate crawl for each of the three attacks, in total we visited 900,000 pages. All measurements were taken between June 2017 and December 2018 on Amazon EC2 servers located in the U.S..

3.2 The bait technique

Our core measurement contribution is the development of a *bait* technique, which allows us to inject sensitive user data into the context of real websites in such a way that third-party scripts can access and exfiltrate the data. We program OpenWPM, our measurement tool, to *behave as if it has already interacted with the site*, instead of simulating all possible interactions of a real user with a site—which would require us to perform tasks that are difficult to automate, such as registering for an account. The bait technique takes different forms depending on the measurement. To measure login manager misuse, we use Firefox's `nsILoginManager` [22] interface to automatically add an email address and password pair as if the user has already saved those credentials for each page we visit (Section 4). To measure social data exfiltration, we simulate a user who logged in to Facebook by replicating the Facebook SDK's interface (Section 5). Finally, to measure Whole-DOM exfiltration, we directly inject PII (email address, name, and surname) to the page DOM (Section 6).

Our work is conceptually similar to past studies that have detected privacy leaks to third parties in real user data [11, 23], but our decision to simulate user data has several distinct advantages. By simulating user data we are able to collect detailed measurements without the risk of compromising user privacy. Further, not having real users in the loop allows us to run our measurements at a large scale. A consequence of this is that we may bait sites in ways that a real user interaction would never create, such as saving user credentials on a site that has no account support. Therefore, although our method allows us to automatically identify third parties that have exfiltration functionality, we need manual verification to claim that a third party exfiltrates sensitive information on a specific site.

The bait technique has three core components: (1) injecting sensitive data into the page context, (2) monitoring access to the data and attributing access to a specific third party, and (3) detecting transmission of the data to a third-party server. We extend OpenWPM to carry out the first two components, and provide details on how we do so in the sections below. The third component, data exfiltration, is detected using the method detailed in Section 3.4.

3.3 Instrumentation

We extended OpenWPM's already existing JavaScript instrumentation to capture access to a set of DOM properties and function calls that can be relevant for detecting the attacks. To capture JavaScript stack traces, OpenWPM throws and catches an Error when an instrumented property is accessed, or a monitored function is invoked [2]. OpenWPM then reads the stack trace from the "stack" property of the caught Error [1]. We extended OpenWPM by adding support to capture JavaScript stack traces for HTTP requests, which correspond to the "Stack trace" column of Firefox's Network monitor. Capturing stack traces for HTTP requests required a separate mechanism that we built on top of the existing HTTP instrumentation [25]. Specifically, for each HTTP request, we recorded the snapshot of the current JavaScript call stack using the `Components.stack` XPCOM language binding [26]. The recorded JavaScript call stacks are composed of stack frames that contain the calling code's file name (URL), function name, line and column number. Recording call stacks for requests allowed us to attribute data exfiltrations with the granularity of script line and column numbers.

Using our extensive JavaScript instrumentation, we record script behavior that can be a sign of data exfiltration. To that end, we record function calls and property accesses including registering listeners for mouse and form events (e.g. `blur`, `keypress`, `keyup`), reading filled out form data, and triggering HTTP requests to third-party servers.

Additional instrumentation specific to each attack is described in the *Measurement methods* part of each attack section (i.e., Sections 4, 5 and 6).

3.4 Detecting and attributing data leakage

To detect leaks, we build on the method proposed by Englehardt et al. [27]. The method searches for encoded and hashed versions of PII that the crawler injects into pages. We configure our heuristics to detect up to three layers of encoding and hashing from a set of ten encoding and 16 hash functions such as MD5, SHA-1 and SHA-256. The full set of supported encodings and hash functions is given in Appendix C. We search for the username part of the email address (in addition to the full email address) to detect leaks by parties who send username and domain of the email addresses separately.

We searched URLs, `Cookie` headers, `Referer` headers of both GET and POST requests. In addition, we searched the POST request bodies. We exclude first-party HTTP requests and responses from our analysis. Some of the crawled pages redirected to addresses that are not under the same domain (PS+1) as the original address. For instance many login links collected from different sites redirect to Twitter's oAuth page. We exclude these visits from our analysis.

4 Login manager misuse

All major browsers have built-in login managers that save and automatically fill in credentials to make the login experience more seamless. The set of heuristics used to determine which login forms will be autofilled varies by browser, but the basic requirement is that a username and password field be available.

The underlying vulnerability of login managers to credential theft has been known for years. Much of the past discussion has focused on password exfiltration by malicious scripts through cross-site scripting (XSS) attacks [28, 29].

Mechanism. Figure 1 details the attack. First, a user fills out a login form on the page and asks the browser to save the login credentials. The third-party script does not need to be present on the login page. Then, the user visits another page on the same website which includes the third-party script. The script inserts an invisible login form, which is automatically filled in by the browser's login manager. The third-party script retrieves the user's email address by reading the populated form and sends the email hashes to third-party servers.

Measurement methods. To study password manager abuse, we extended OpenWPM to simulate a user with saved login credentials and added instrumentation to monitor form access. We used Firefox's `nsILoginManager` [22] interface to add credentials as if they were previously stored by the user. We did not otherwise alter the functionality of the password manager or attempt to fill login forms.

1 This method was adapted from the EFF's Privacy Badger Extension [24]

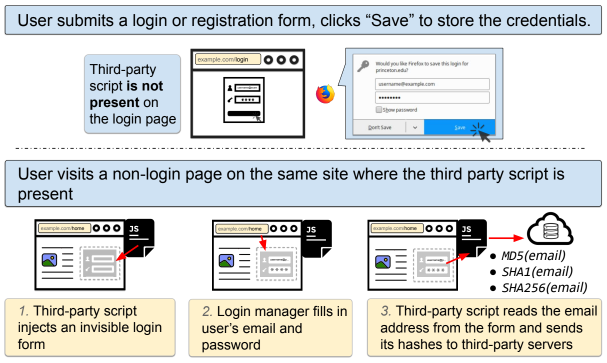
User submits a login or registration form, clicks "Save" to store the credentials.
Third-party script is not present on the login page
User visits a non-login page on the same site where the third party script is present
MD5(email)
SHA1(email)
SHA256(email)
1. Third-party script injects an invisible login form
2. Login manager fills in user's email and password
3. Third-party script reads the email address from the form and sends its hashes to third-party servers

Fig. 1. The process by which a third-party script can extract a user's credentials from the browser's login manager.

The fake credentials acted as bait (Section 3.2). To detect when the credentials were accessed, we added the following probes to OpenWPM's JavaScript monitoring instrumentation (Section 3.3):

1. Monitor the DOM mutation event `DOMNodeInserted` to detect the injection of a new login form or input elements into the DOM. When this event fires, we serialize and log the inserted HTML elements.
2. Instrument `HTMLInputElement` and `HTMLFormElement` objects to intercept access to form input fields. We log the input field value that is being read to detect when the bait email (autofilled by the built-in login manager) was read.
3. Store HTTP request and responses to detect when the username or password is sent to a remote server using the leak detection method detailed in Section 3.4.

For both the JavaScript (1, 2) and the HTTP (3) instrumentation we store JavaScript stack traces at the time of the function call or the HTTP request. We parse the stack trace to pin down the initiators of the HTTP request or the parties responsible for inserting and accessing a form. Specifically, we search for scripts that do all of the following:

– Inject an invisible HTML element containing a password field
– Read the email address from an input field that was automatically filled by the browser's login manager
– Send the email address, or a hash of it, to a third-party server

Results. Applying this selection criteria to our measurement data we found two scripts, loaded from AdThink (audienceinsights.net) and OnAudience (behavioralengine.com), which used this technique to extract email addresses from login managers. We also reproduced the email address collection by manually registering for websites that embedded these two scripts and allowed the browser to save the credentials in the process. We summarize our findings in Table 1.

Company	Script address	# sites	# sites (top 1M)
Adthink	static.audienceinsights.net/t.js	9	1047
OnAudience	api.behavioralengine.com/scripts/be-init.js	7	63

Table 1. We found two scripts that misused the browser login manager to extract user email addresses. These scripts were found on 1,110 of the Alexa top 1 million sites in the September 2017 Princeton Web Census crawl [2, 30].

We provide code snippets from the two scripts in Listing 1 and 2. Adthink's script sent the MD5, SHA1 and SHA256 hashes of the email address to its server (secure.audienceinsights.net), as well as the MD5 hash of the email address to the data broker Acxiom (p-eu.acxiom-online.com). OnAudience's script sent the MD5 hash of the email back to its own server. In addition, OnAudience's script collected browser features including plugins, MIME types, screen dimensions, language, timezone information, user agent string, OS and CPU information. OnAudience's script was most commonly present on Polish websites, including newspapers, ISPs and online retailers. 45 of the 63 sites that embedded OnAudience's script at the time of measurement had the ".pl" country code top-level domain.

We also discovered that autofilled credentials were leaked to third parties on an additional 121 sites where neither of the two scripts were present. Analyzing such cases we determined that those leaks happen due to session recording and identity tracking scripts, which indiscriminately monitor and exfiltrate form data (Section 6). On those sites, login forms are inserted by first-party scripts or other authentication related scripts for non-tracking purposes.

At the time of measurement, login form autofilling did not require user interaction; all of the major browsers would autofill the username (often an email address) immediately, regardless of the visibility of the form. Chrome did not autofill the password field until the user clicks or touches anywhere on the page. The other browsers we tested did not require user interaction to autofill password fields.

5 Social data exfiltration

In this section we investigate the exfiltration of personal identifiers from websites through "login with Facebook" social API.

Mechanism. Facebook Login and other social login systems simplify the account creation process for users by decreasing the number of passwords to remember. But social login brings risks: Cambridge Analytica was found misusing user data collected by a Facebook quiz app which used the Facebook Login service [31, 32]. We have uncovered an additional risk: when a user grants a website access to their social media profile, they are not only trusting that website, but also third parties embedded on that site.

When third parties are embedded into a first-party execution context in which the user has approved a Facebook login integration, the third-party script will be able to query the Facebook API with the privileges and access of the first-party. This can allow the third-party script to query any user account information that the user has granted the first-party access to, which includes the user's ID. The user ID collected through the Facebook API is specific to the website (or the "application" in Facebook's terminology), which would limit the potential for cross-site tracking. However, at the time of our measurements it was possible to use these app-scoped user IDs to retrieve the global Facebook ID, user's profile photo, and other public profile information, which can be used to identify and track users across websites and devices [33, 34].

Measurement methods. To study the abuse of social login APIs we extended OpenWPM to simulate that the user has authenticated and given full permissions to the Facebook Login SDK on all sites. To spoof that a user is logged in, we created our own `window.FB` object and replicated the interface of version 2.8 of the Facebook SDK. The spoofed API had the following properties:

For method calls that normally return personal information we spoofed the return values as if the user is logged in and called the callback functions with necessary arguments. These included `FB.api()`, `FB.init()`, `FB.getLoginStatus()`, `FB.Event.subscribe()` for the events `auth.login`, `auth.authResponseChange`, and `auth.statusChange`, and `FB.getAuthResponse()`. For the Graph API (`FB.api`), we supported most of the profile data fields supported by the real Facebook SDK. We parsed the requested fields and return a data object in the same format the real graph API would return. For method calls that do not return personal information we simply called a no-op function and ignored any callback arguments. This helped minimize breakage if a site calls a method we do not fully replicate. We fired `window.fbAsyncInit` once the document has finished loading. This callback function is normally called by the Facebook SDK during the initialization. The spoofed `window.FB` object is injected into every frame on every page load, regardless of the presence of a real Facebook SDK. We then monitored access to the API using OpenWPM's JavaScript call monitoring. All HTTP request and response data, including HTTP POST payloads are examined to detect the exfiltration of any of the spoofed profile data (including that which has been hashed or encoded). Since we did not inject the user's identity into the page in this experiment, any personal data accessed and leaked by scripts must have been queried from our spoofed API.

We stored the JavaScript stack traces for calls to `window.FB` and HTTP requests. We then used these stack traces to determine which APIs scripts accessed and when they were sending data back. For some scripts our instrumentation only captured the API access, but not the exfiltration. In these cases, we manually debugged the scripts to determine whether the data was only used locally or if it was obfuscated before being transmitted. We explicitly note the cases where we could not make this determination.

Results. We found seven loosely-integrated[2] third-party scripts collecting Facebook user data using the first party's Facebook access (Table 2). These scripts were embedded on a total of 434 of the top 1 million sites at the time of measurement[3]. Six of the seven scripts collected the user ID, and two scripts collected additional profile information such as email and username.

Our leak detection method did not detect leaks by four of the seven scripts (Augur, ProPS, Tealium, Forter) displayed in Table 2. We used manual analysis to check for social API leaks by these four scripts, and

2 We use the following criteria to determine how tightly integrated a third-party script is with a first party: 1) whether the third-party script initiates the Facebook login process instead of passively waiting for the login to happen and 2) whether the third-party script contains the Facebook App ID of the website it is embedded on. The scripts summarized in this section neither initiated the login process, nor contained the App ID of the websites.

3 See, https://gist.github.com/englehardt/bf976e6b90f44f735749014a7a4a48b6

Company	Script Domain	Facebook Data Collected	Leak detected via
OnAudience	api.behavioralengine.com	User ID (hashed), Email (hashed), Gender	Automated analysis
Augur	cdn.augur.io	Email, Username	Manual analysis
Lytics*	d3c…psk.cloudfront.net	User ID	Automated analysis
Nativka	p1.ntvk1.ru	User ID	Automated analysis
ProPS	st-a.props.id	User ID	Unable to verify leak
Tealium	tags.tiqcdn.com	User ID	Unable to verify leak
Forter	cdn4.forter.com	User ID	Unable to verify leak

Table 2. The list of third-party scripts collecting Facebook user data using the first party's Facebook access at the time of measurement.
*: The script loaded by Opentag tag manager included a code snippet which accessed the Facebook API and sent the user ID to an endpoint of the form https://c.lytics.io/c/1299?fbstatus=[...]&fbuid=[...]&[...] This snippet appears to be a modified version of an example code snippet found on the lytics.github.io website with the description "Capturing Facebook Events". The example code appears to provide instructions for first parties to collect Facebook user ID and login status.

found leaks by one of them (Augur). Specifically, we used a combination of the JavaScript Deobfuscator browser extension [35] and devtools to determine how the data was used after being collected from the API. We limited the manual analysis to scripts that send at least one request after they access the Facebook API. For ProPS, Tealium and Forter, we were unable to manually verify that the social API data was transferred to a remote server.

While we cannot say how these third parties use the information they collect, or whether they have been acting on behalf of the first party, we examined their marketing material at the time of measurement to understand how they might have been used. OnAudience, Tealium AudienceStream, Lytics, and ProPS all offered some form of "customer data platform", which collects data to help publishers to better monetize their users. Forter offered "identity-based fraud prevention" for e-commerce sites. Augur offered cross-device tracking and consumer recognition services. Nativka offered traffic growth and content monetization services.

In Appendix B we present a related tracking vulnerability we discovered and responsibly disclosed using these measurement methods. In particular, we found a third party using its own social login in embedded iframes in a way that allowed their users to be tracked by other parties.

In this study we focused on Facebook Login as it is the most widely used social SDK on the web [36], but the vulnerabilities we described are likely to exist for most social login providers. Indeed, Forter and Nativka, two scripts listed in Table 2, appear to access user identifiers from the Google Plus API (Forter) [4], and from the Russian social media site VK's Open API (Nativka) [5].

6 Whole-DOM exfiltration

In this section we examine third-party scripts that collect the full contents of the DOM (Document Object Model, the tree of objects that constitutes the web page), as well as those that monitor all mouse movements and key presses on the page. Scripts may collect this data to provide analytics services based on users' clicks and other interactions on the page.

Mechanism. The top-level context's DOM can contain sensitive information. When a user logs in to a site, the site may display her name or email address in the text of the page. Similarly, users may enter personal information, such as their address, credit card number, or social security number into forms on the page. For some sites this information may be even more sensitive, such as the user's bank account balance or medical history. Third-party scripts embedded in the top-level context have access to the same information that is displayed to the user when she visits the site. Malicious scripts can abuse this access to surreptitiously collect user information. However, this sensitive information can also get scooped up by benign scripts which collect portions of the DOM as part of the services they provide to first parties.

There are a number of ways a script can grab the contents of the DOM, including: reading `outerHTML`, `innerHTML`, or the `textContent` property of `document.body`, looping through all elements in the DOM and serializing them individually. To monitor user interactions, scripts can use event listeners for mouse and keyboard events. Finally, scripts can listen to the `blur` or `change` events, which are fired when a user interacts with input elements on the page.

Detection Method. We took a two-step approach to detecting whole DOM exfiltration. First, we appended several unique *bait* values to the DOM of all frames present on a page and search for these values in

4 See, https://gist.github.com/englehardt/9801bef634dc29699116a489f33d850b
5 See, https://gist.github.com/englehardt/65d2959be03bfafdc1c25d57965f4539

the resulting network traffic. The values are added by creating a new `div` element and adding it directly to `document.body`. The `div` had the style `display:none` set to prevent it from altering the layout of the page. The added values included an email address and the string `Welcome <FirstName> <LastName>!`, where `<FirstName>` and `<LastName>` are unique strings. We chose to include bait values that match the format of a real users PII to detect if any scripts are parsing information out of the DOM. We used the HTTP stack traces and the leak detection method detailed in Section 3.4 to discover and attribute leaks of the injected email address or name.

During our preliminary analysis, we discovered several instances where scripts compress and split the page source over multiple requests, which our standard leak detection method fails to detect. To capture these instances, we took a different approach:

1. **Determine which pages contain scripts that might be exfiltrating the DOM.** For each page, we summed the total size of data sent to each third party in POST requests on that page. We then generated a list of pages where at least one third party receives a total amount of data larger than the compressed length of the page text[6].
2. **Re-measure the candidate sites with and without a large chunk of data appended to the DOM.** For all sites on which we suspect third-party DOM exfiltration, we re-measured the site twice: once with a 200 Kilobyte chunk appended directly to the `body` element of the DOM, and once without any data appended to the DOM. We chose 200 Kilobytes as the chunk size because it was large enough to outweigh small differences in page size between the crawls (such as a changing headline), but not so large as to disrupt services that collected data from the DOM.
3. **Measure the difference in payload size for each third party between the crawls.** We summed the total payload size across all POST requests which occur during a single page visit for each third party. We then compared the total POST request size between the two crawls, and flagged any third-party script that had a difference greater than the compressed length of the injected chunk of data (approximately 150 Kilobytes).

The two detection methods described above provided a list of scripts which appear to collect data from the DOM. To better understand how the scripts are monitoring interaction with the DOM we used OpenWPM's JavaScript call monitoring (Section 3.3) instrumentation to record the following: calls to `innerHTML`, `outerHTML`, `innerText`, and `outerText` on the `HTMLBodyElement` and the `documentElement`. In addition we observed all event listener registrations on the `HTMLBodyElement`, the `window.document` object, and the `window` object. We examined registrations which monitor events that capture the user's mouse movements, page interactions, and key presses.[7] We took a script's use of these events as a signal that they are monitoring user interaction with the DOM in addition to scraping the contents of the DOM.

Finally, we used a combination of the presence of DOM scraping, the registration of event handlers monitoring user interaction, and a manual examination of the marketing materials of the companies involved to determine the type of services the third-party script offers. We manually reviewed the third party's website and product offerings in early 2018. Scripts that offered ad injection, affiliate link insertion, or content monetization were considered "Advertising", scripts that offered session recordings and customer insight tools were considered "Analytics", scripts that offered custom support tools were considered "Support", and scripts that offered translation services were considered "Translation". The results of this analysis are summarized below.

Results. We found no instances of malicious scripts parsing the DOM to exfiltrate user data. However, we did find a number of companies doing full-page scraping, collecting all of the text on the page or serializing portions of the DOM. Although these services did not appear to be built with the intention of collecting PII, the broad nature of the collection techniques made it very easy for PII to get scooped up with the rest of the collected data. We summarize our findings in Table 3. The majority of the scripts we discovered sent requests which included the name and email address inserted by our instrumentation. Eight of the 28 scripts were only discovered by our chunk injection measurement due to unsupported payload encodings. In all cases, the data was transferred to a third-party collection endpoint via a `POST` request.

6 We used Beautiful Soup [37] and zlib [38] Python libraries to extract and compress the page text, respectively.

7 The event listener registration events that we monitored included: `mouseup`, `mousedown`, `click`, `auxclick`, `mousemove`, `wheel`, `dblclick`, `select`, `contentmenu`, `mouseleave`, `mouseout`, `mouseenter`, `mouseover`, `keydown`, `keyup`, `keypress`, and `scroll`.

Service	Purpose	# sites
Yandex Metrika	Analytics	198
FullStory	Analytics	55
Hotjar	Analytics	48
SkimLinks	Advertising	34
Sessioncam	Analytics	18
UserReplay	Analytics	15
Transifex	Translation	9
VWO	Analytics	7
Tealeaf	Analytics	7
Jornaya	Analytics	5
IntelliTXT	Advertising	4
Digidip	Advertising	4
RedLink	Analytics	3
Localizer	Translation	2
Viglink	Advertising	2
Prosperent	Advertising	1
Wovn	Translation	1
xclaimwords	Unknown	1
Bkred.ru	Unknown	1
ABTasty	Analytics	1

(a) Services detected by both ID injection and chunk injection.

Service	Purpose	# sites
Clicktale	Analytics	37
Smartlook	Analytics	31
Lucky Orange	Analytics	23
Quantum Metric	Analytics	11
Inspectlet	Analytics	10
Mouseflow	Analytics	5
LogRocket	Analytics	2
SaleMove	Support	1

(b) Services detected only by chunk injection. Since our chunk injection measurement was run on a sample of the 50,000 sites measured in (a), we expect the site counts for these services to be underrepresented relative to (a).

Table 3. The top companies that we discovered scraping information from the DOM at the time of our June or November 2017 measurements. The apparent purpose of data collection includes: **Analytics:** heatmaps, session replay, form analytics, **Advertising:** mouse-over keyword ads, automatic affiliate link insertion, **Translation:** automatic localization, and **Support:** live customer support. Scripts grab either the text on the page or a representation of the DOM, which can range from a complete serialization to a custom encoding of some elements.

The majority of the companies collect DOM data to provide analytics services to the first party. Of these, the most commonly provided service is "session replay", which was offered by 16 companies at the time of measurement. Session replay services allow the first party to observe how their page was rendered for the user and how the user interacted with their site. All of the session replay providers collected some custom encoding of all nodes and text in the DOM, in some cases including all of the inline script and CSS content. We believe this is necessary to allow the recording to accurately reflect the experience of the user, given that many pages are built dynamically and may change from user to user.

In the remaining cases, text was extracted from the DOM and sent to a third party. While we are not able to measure exactly *how* the text data is used, we studied the marketing material of the companies to better understand how it might be used. Several of the companies provided automatic monetization of product or retailer references by attaching affiliate links to specific keywords. For example, a reference to a new model of Nike shoes would be replaced by an affiliate link to a store where the user can purchase that shoe. IntelliTXT's advertising product is slight variation of this; rather than replacing the text with an affiliate link, it replaces the text with a mouse-over advertisement that displays in a tooltip next to the text. Finally, three of the companies offer translation services, which include automatic localization of site content.

We found no evidence that suggests the scraped data was used for advertisement personalization or cross-site tracking by any of the companies analyzed. In fact, several of the analytics companies explicitly forbid the collection of sensitive user information using their services [39, 40], and provide automated and manual features that first parties can use to prevent the collection of sensitive user data. As an example, we observed code in VigLink and Wovn scripts which filter the contents of the collected data using regular expressions. VigLink's filtering appears to be motivated by a desire to protect user privacy, as evidenced by the inclusion of references to `pii`. Their script prevented the collection of email addresses and strings of integers between 6 and 18 characters in length. Wovn's filtering appeared to be intended to prevent the collection of non-translatable strings and included email addresses.

6.1 Case study: the ineffectiveness of session replay redaction tools

To better understand the effectiveness of the data privacy features offered by third-party services, we performed an in-depth examination of the redaction tools provided by six of the companies offering session replay services: FullStory, UserReplay, SessionCam, Hotjar, Yandex, and Smartlook. Session replay analytics are meant to provide insights into how users interact with websites and discovering broken or confusing pages. However the extent of data collected by these services is likely to exceed user expectations; text typed into forms is collected before the user submits the form, and precise mouse movements are saved, all without any visual indication to the user.

The replay services offer a combination of manual and automatic redaction tools that allow publishers to exclude sensitive information from recordings. However, in order for leaks to be avoided, publishers would need to diligently check and scrub all pages which display or accept user information. For dynamically generated sites, this process would involve inspecting the underlying web application's server-side code. Further, this process would need to be repeated every time a site is updated or the web application that powers the site is changed.

Redacted Field	FullStory	UserReplay	SessionCam	Hotjar	Yandex	Smartlook
Name	○	◐	◐	○	○	○
Email	○	◐	◐	○	○	○
Phone	○	◐	◐	○	○	○
Address	○	◐	◐	○†	○	○
SSN	○	◐	◐	○	○	○
DOB	○	◐	◐	○	○	○
Password	●	◐	●	●	●	◐
CC Number	●	◐‡	◐	◐	○	●
CC CVC	●	◐	◐	○	○	●
CC Expiry	●	◐	◐	○	○	●

Table 4. Summary of the automated redaction features for form inputs enabled by default from each company at the time of measurement in November 2017.
●: Data is excluded; ◐: equivalent length masking; ○: Data is sent in the clear. † Hotjar masked the street address portion of the address field. ‡: UserReplay sent the last 4 digits of the credit card field in plain text.

To better understand the effectiveness of these redaction practices, we set up test pages during November 2017 and installed replay scripts from the six companies. All of the companies studied offered some mitigation through automated redaction, but the coverage offered varied greatly by provider. UserReplay and SessionCam replaced all user input with an equivalent length masking text, while FullStory, Hotjar, and Smartlook excluded specific input fields by type. We summarize the redaction of other fields in Table 4.

Automated redaction is imperfect; fields are redacted by input element type or heuristics, which may not always match the implementation used by publishers. For example, FullStory redacted credit card fields with the `autocomplete` attribute set to `cc-number`, but collected any credit card numbers included in forms without this attribute. Indeed, we discovered credit card data leaking to FullStory from input fields that lacked `autocomplete` attributes (see Table 5).

To supplement automated redaction, several of the session replay companies, including Smartlook, Yandex, FullStory, SessionCam, and Hotjar allowed sites to further specify input elements to be excluded from the recording. To effectively deploy these mitigations a publisher would need to actively audit every input element to determine if it contains personal data. A safer approach would have been to mask or redact all inputs by default, as was done by UserReplay and SessionCam, and allow whitelisting of known-safe values. Even fully masked inputs provide imperfect protection. For example, the masking used by UserReplay and Smartlook at the time of measurement leaked the length of the user's password.

Several of the session replay companies also offered redaction options for displayed content, i.e. content that would be collected through scraping the DOM. Unlike user input recording, none of the companies appeared to provide automated redaction of displayed content by default; all displayed content on our test page ended up leaking. Instead, session replay companies expect sites to manually redact all personally identifying information included in the DOM. Sensitive user data has a number of avenues to end up in recordings, and small leaks over several pages can lead to a large accumulation of personal data in a single session recording.

To understand how well the manual redaction features work in practice, we manually examined around 50 of the top sites on which we found session replay scripts. We discovered several categories of sensitive information leaks during our interactions, including: passwords, medical information, student data, credit card data, and purchase information. Table 5 summarizes our findings.

We observed password leaks on three of the surveyed websites. On two of the sites, propellerads.com and johnlewis.com, the password leak was caused by the way the sites implemented a "show password" feature. In both instances, the sites stored the user's password in two input elements: one of type `password` and one of type `text`. When the user interacts with the "show password" feature, the sites would swap the two input elements, causing the user's password to become visible. Both FullStory's and SessionCam's automated redaction rules failed to capture the input element of type `text`[8]. In the third case, the password leak was caused by a bug in the way FullStory's manual redaction feature applied to input fields of type `password`. We disclosed the password leaks to respective first and third parties, and we were informed by the third party services that the issues were later fixed (Section 7).

With the exception of walgreens.com, the remainder of the leaks largely appeared to be caused by a sparse use of redaction on those pages. Walgreens made extensive use of display content redaction, but the user's name and prescription choices appeared on subsequent pages of the checkout process. Similarly, the identity verification page asked several multiple choice questions con-

8 Browser extensions such as Show Password [41] and Unmask Password [42] implement the same "show password" feature by swapping the password field with a cleartext field. We found that on certain websites including Lenovo.com, FullStory will inadvertently collect passwords when the Show Password Chrome extension is in use.

First party	Third party	Data Leaked	Cause
walgreens.com	FullStory	prescriptions, name, identity	unredacted display data
propellerads.com	FullStory	passwords	"show password" feature
johnlewis.com	SessionCam	passwords	"show password" feature
wpengine.com	FullStory	passwords	bug in FullStory redaction
gradescope.com*	FullStory	student data	unredacted display data
lenovo.com	FullStory	billing information	unredacted display data
bonobos.com	FullStory	credit card data	unredacted input data

Table 5. A sample of sensitive data leaks to session replay companies that we observed during a manual inspection of sites between November 2017 and February 2018.
*: We discovered the leak on Gradescope thanks to an external tip.

taining sensitive user data—the radio buttons for the questions were redacted from recordings, but the mouse traces would still reveal the user's answers.

To recap, we identify three main causes of unintended collection of sensitive data by session replay scripts:

1. **Automated redaction heuristics that are based on unrealistic assumptions**: Automated redaction heuristics make assumptions about input elements' type and attributes, but websites may implement an interface (e.g. a password box) in different ways that these assumptions may not cover. For example, the assumption that passwords are only stored in input fields of type `password` did not hold on propellerads.com and johnlewis.com, which caused passwords to leak on those sites.
2. **Manual redaction rules that fail to cover all sensitive data**: Building and maintaining manual redaction rules that cover all sensitive data can be an error-prone task especially for complex and dynamic websites. Leaks due to unredacted student data on gradescope.com, and billing information on lenovo.com are two examples of this type of failure.
3. **Interactions between first and third-party scripts, and browser extensions** affect both automated redaction heuristics and manual redaction. It is difficult for publishers and third parties to foresee all possible interactions between different scripts. The password leak we identified on lenovo.com (when the Show Password Chrome extension is in use) shows even the extensions installed on users' browsers need to be taken into account. Since extensions may modify the DOM in arbitrary ways, anticipating all potential failures is not feasible.

7 A retrospective look

We first publicly released the findings reported here in 2017-18 (whenever possible, we notified affected parties before public disclosure) [54, 58–61]. This gives us an unusual opportunity for a retrospective look at the effects of a large-scale analysis of privacy vulnerabilities. We find that the specific vulnerabilities we reported have largely been addressed; however, the root causes of the problem remain intact, with the likelihood that new vulnerabilities are regularly being created. Based on this analysis, we suggest two potential paths forward.

7.1 Review of responses to our findings

The publication of our findings resulted in several privacy fixes and improvements to browsers, websites, third parties and privacy protection tools. Table 6 provides an overview. In this section we summarize the changes that resulted from our work[9], and examine how the fixes—while improving the status quo—often fail to address the underlying vulnerabilities.

Login Manager misuse. Adthink and OnAudience[10] stopped misusing browser login managers to extract user email addresses following the publication of our results [52, 53]. Brave and Safari[11] disabled automatic login manager credential filling, and instead require that the user first interact with the password field [46, 47]. Our work also inspired Mozilla and Chrome to reconsider implementing similar changes, which had been previously proposed [43, 44]. Publishers

9 We determine whether a fix was deployed as a response to our initial publications based on the references provided in the bug descriptions or commit messages [43, 44, 47, 49, 50], publications and quotes from companies in press articles covering our preliminary publications [52, 55–57], and from personal communications we received as a response to our disclosures [54, 58]. We explicitly note the cases where the fixes were deployed immediately following our initial publications without an explicit reference to our findings [46, 51, 53].

10 Immediately following our initial publication [53], OnAudience removed the part of their script that misuses browsers' login manager without referring to our findings.

11 The related CVE [62] involving Safari Login AutoFill was created five days after the publication of our preliminary findings on login manager misuse [59] without a reference to our findings. The vulnerability is described as follows: "It allows remote attackers to read autofilled data by leveraging lack of a user-confirmation requirement."

Party		Attacks		
		Login Manager	Social Integration	DOM Exfiltration
Browsers	Chrome	Considering restrictions [43]	No fix	No Fix
	Firefox	Considering restrictions [44]	Proposal [45]	Proposal [45]
	Safari	Require interaction [46]	No fix	No fix
	Brave	Require interaction [47]	Proposal [48]	Proposal [48]
Blocklists	EL/EP*	Already Blocked	Not blocked†	Blocked [49]
	Disconnect	Blocked [50]	Not blocked‡	Blocked [51]
Third parties	Adthink	Stopped [52]	N/A	N/A
	OnAudience	Stopped [53]	Stopped	N/A
	FullStory	N/A	N/A	Fixed PW leak bug [54]
	Facebook	N/A	N/A	Limited [55]
	Smartlook	N/A	N/A	Limited [56]
	Yandex	N/A	N/A	Limited [56]
First Parties	Walgreens	N/A	N/A	Stopped [57]
	Bonobos	N/A	N/A	Stopped [57]
	Gradescope	N/A	N/A	Stopped [58]

Table 6. A summary of deployed fixes by browsers, blocklists, and the first and third parties we studied. For the DOM exfiltration attacks we also include two browser proposals that are motivated by the types of attacks examined in this work.
* EL/EP: EasyList and EasyPrivacy. †: No third parties were added to EL/EP due to Social data exfiltration. EL/EP already blocked three of the seven scripts from Table 2 (ProPS, OnAudience, Nativka) at the time of our measurement and blocked Forter since, but not as a response to our study. ‡: Disconnect blocked behavioralengine.com due to its Login manager misuse [50]. Augur was already blocked. Remaining five scripts in Table 2 are not blocked.

can achieve a similar level of protection in all browsers by isolating login forms on a separate origin (e.g., a subdomain of their main domain), which will prevent most browsers from filling credentials on non-login pages.

These changes reduce the attack surface available to third parties wishing to exfiltrate user credentials; a user's credentials are no longer available on pages other than the login page. However, third parties embedded on a site's login page can continue to extract the user's credentials after the user enters them manually or interacts with the browser's credential manager.

Social data exfiltration. Facebook disabled the feature to resolve app-scoped user IDs to global IDs [55][12]. This limited the ability of a third party to collect Facebook's global identifier for the user through their SDK. However, third parties integrated into the main context of a page can continue to query user data from integrated social media widgets with the same privileges as the first party. To prevent this vulnerability entirely, first parties must integrate social media widgets on a separate origin that doesn't embed any third-party content. This can greatly increase the engineering complexity of a website, and can limit how tightly integrated the social media content is with the rest of the user experience.

Whole DOM exfiltration. Walgreens, Bonobos, Gradescope stopped using session replay scripts [57, 58]. This fixed the leaks discovered during our manual review, but our 50 site review represents a fraction of the total number of websites that embed these services (99,173 in the Alexa top 1 million, based on the data from the September 2017 Princeton Web Census crawl [2, 30]). Similarly, FullStory fixed the bug that allowed passwords to leak on WPEngine.com or sites with a similar markup [54]. Finally, Smartlook and Yan-

12 We received a communication from Facebook indicating that the fix was deployed as a response to our publication.

dex committed to prioritizing HTTPS dashboards for replaying session records [56] [13].

There are a number of steps both first parties and third parties can take to mitigate some of the risk introduced by these services. Rather than rely on redaction tools, publishers can restrict third-party replay services to pages that do not contain any sensitive user data. Session replay companies can restrict their tools to recording basic information about mouse movements and keypresses, and replay those events over DOM data captured from a public version of the page. Likewise, session replay companies can restrict their collection to keypress *events*, rather then recording actual key values. These mitigations are imperfect from a privacy perspective; they also compromise functionality.

Blocklists. Users can protect against all three of these types of attacks by installing privacy tools that block tracking resources. The domains we found to misuse browser login managers were already blocked by the EasyPrivacy blocklist at the time of measurement, and were later added to Disconnect's blocklist [50]. Similarly, many of the domains used to serve the session replay scripts were already blocked by EasyPrivacy at the time of our measurements, and several more—including domains from FullStory, Smartlook, and UserReplay—were added after we released our results [49]. Likewise, Disconnect added several new session replay scripts to their lists after we released our results [51][14].

While these results highlight the effectiveness of measurement and blocking, the types of analyses described in this paper would need to be regularly re-run to keep the list of blocked resources up-to-date. To the best of our knowledge, these measurements have not been repeated in the two years since we released our results. There are a number of challenges facing repeated measurements, which we further explore in Section 7.2.

7.2 Root causes of failures

To recap: the publication of our findings led to a number of fixes, but we have shown how these fixes are fundamentally reactive—the specific problems that we found were (mostly) fixed, but it remains possible for similar but new problems to be introduced. Two years after our initial findings, there have not been any comprehensive fixes deployed for these issues. We identify three fundamental challenges.

SOP is all-or-nothing. The Same Origin Policy rests on the assumption that a first party trusts a third party fully or not at all. Unfortunately, this model does not capture the range of trust relationships that we see on the web. Since many of these third parties cannot provide their services if they are isolated, the first parties have no choice but to give them full privileges even though they do not trust them fully.

Transitivity of trust. A user may trust a website and the website may trust a third party, but the user may not trust the third party. Trust becomes even less transitive with longer redirect chains. The web's security model assumes transitive trust.

Economics. Many third parties advertise the ease of deploying their scripts and the fact that no technical expertise is necessary. For example, FullStory's web page says: "Set-up is a thing of beauty. To get started, place one small snippet of code on your site. That's it. Forever. Seriously." [63]. Based on our interactions with publishers, we believe that this is a key reason for the popularity of these third parties. The small publishers that rely on them lack the budget to hire technical experts internally to replicate their functionality. In fact, even the careful review of DOM elements needed for proper redaction ends up being a significant burden to these first parties, and hence omitted.

Economic considerations may help explain the limited adoption of technical solutions such as Caja [64] or ConScript [65] that better capture the partial trust relationship between first and third parties. Unfortunately, such solutions require the first party developer to reason carefully about the privileges and capabilities provided to each individual third party. We believe that if first parties possessed this expertise, they would have applied it to ensure proper configuration of existing tools, and our study would not have found the widespread leaks that it did.

7.3 Two potential paths forward

The two classic information security paradigms are the reactive approach and the preventive approach. Both of them provide potential paths to addressing the privacy problems of interest to us, but each approach has its challenges and limitations.

13 We have verified that Smartlook and Yandex switched to HTTPS dashboards for replaying session recordings.

14 The commit message for this update mentions "screen tracking services" and adds third parties we detected to their blocklist one day after we publish our blog post, but it does not refer to our findings.

The **reactive approach** relies on early detection and mitigation of vulnerabilities. Our work provides evidence that large-scale detection of PII leaks is possible and that specific vulnerabilities will be addressed quickly once discovered.

Our work also shows that large-scale detection is technically challenging, necessitating the time investment of experts. Our measurement tools require constant maintenance to keep up with browser updates and interface changes. Similarly, our measurement methods require manual tuning to respond to adversarial evasion (i.e., false negatives) and changes to benign scripts (i.e., false positives). The data analysis is also not fully automated; manual script inspection and verification will be required for each new measurement.

We are not aware of any efforts to regularly repeat our measurements and, more importantly, look for new classes of vulnerabilities based on the same root causes. Researchers tend not to carry out regular web privacy vulnerability scanning because it does not result in publishable research. Rather, we suggest that browser vendors, privacy advocacy groups, accountability journalists, or some combination thereof should carry out this work.

The **preventive approach** aims to ensure that vulnerabilities do not arise in the first place. As discussed above, fundamental fixes are unlikely to be technical, and even the available technical solutions have been limited in adoption for economic reasons. Thus, we suggest a non-technical preventive approach: regulation that incentivizes first parties to take responsibility for the privacy violations on their websites. This is in contrast to the reactive approach above which essentially shifts the cost of privacy to public-interest groups or other external entities.

Our findings represent potential violations of existing laws, notably the GDPR in the EU and sectoral laws such as HIPAA (healthcare) and FERPA (education) in the United States. However, it is unclear whether the first parties or the third parties would be liable. According to several session replay companies, they are merely data processors and not joint controllers under the GDPR [66–68].[15] In addition, since there are a large number of first parties exposing user data and each violation is relatively small in scope, regulators have not paid much attention to these types of privacy failures. In fact, they risk effectively normalizing such privacy violations due to the lack of enforcement. Thus, either stepped-up enforcement of existing laws or new laws that establish stricter rules could shift incentives, resulting in stronger preventive measures and regular vulnerability scanning by web developers themselves.

15 To the best of our knowledge, this claim has not has been tested in court.

8 Limitations

Our analysis is based on data collected in 2017 and 2018. As explained in Section 7, several browser vendors, third-party libraries and websites changed their behavior in response to the initial publication of our findings. Thus, many of the specific leaks we document are no longer present. Conversely, we chose to study three particular attacks, which is not an exhaustive list of web-based data exfiltration attacks. Despite these limitations, our findings all highlight an underlying problem, namely, loading untrusted JavaScript in a first-party origin.

Many session replay services activate their functionality only for a sample of users, either as explicitly defined by the publisher site or enforced as part of a daily recording limit. Thus, it is possible that our bot that visited the site was not included in the sample, but other users might be.

In Section 5 we observe certain scripts query the Facebook API and retrieve the user's Facebook ID but we could not verify that the ID is sent to their server due to obfuscation of their code and some limitations of our measurement methods such as dealing with obfuscated payloads. We explicitly note these cases in our results.

Although we have released our crawler and instrumentation code, the HTTP request call stack instrumentation we developed is now obsolete due to Firefox's migration from XUL/XPCOM to WebExtensions API [69]. However, OpenWPM's maintainers updated the call stack instrumentation to work with the latest extension API [70] partly based on our work.

OpenWPM's method of mapping property accesses and function calls to script domains may have a limitation when a script uses "eval" to run another script's functions [2]. In addition, OpenWPM may not capture the complete stack traces when scripts inject other scripts or invoke other functions asynchronously (e.g., using `setTimeout`). To avoid misattribution, we complement our automated analysis with manual analysis (Section 4, Section 5), and we base our detection on instrumentation data combined from different sources. This involves intercepting access to form input fields and monitoring DOMNodeInserted events in Section 4, intercepting calls to our spoofed Facebook API in Section 5, and intercepting access to innerHTML, outerHTML, textContent, documentElement, and calls to addEventListener in Section 6.

We use Firefox's HTTP related observer topics such as http-on-modify-request and http-on-examine-response to capture HTTP request and responses. Since at the time of the measurements Firefox did not expose WebSocket and WebRTC connection data through these topics, we might have missed exfiltration attempts using these protocols.

Our measurements were made from US-based EC2 cloud instances using stateless browsers. This may lead to differential treatment; for example, some third parties may not show advertisements to visitors with cloud IP addresses. Moreover, scripts may try to detect OpenWPM, and change their behavior to evade detection. Since each page visit starts with a fresh profile, third parties may attempt to sync cookies on every visit, which may not be the case for real users who keep state in their browsers [71]. In addition, websites may treat European visitors differently, due to stricter privacy regulation. Despite these limitations, we note that a recent study on crawlers' effect on web measurements found that OpenWPM (with bot mitigation feature turned on) results in least amount of server block pages compared to alternatives [72].

9 Conclusion

We highlighted the risks of third party script inclusions by studying login manager abuse, social data exfiltration and whole-DOM exfiltration on 300,000 pages from 50,000 sites. We developed and used a bait technique which allowed us to inject sensitive user data into real websites so that third-party scripts can access and exfiltrate the data. Our findings show that third-party scripts engage in highly invasive practices such as inserting invisible login forms into pages to trigger browsers' login autofill, tapping into social APIs to exfiltrate users' social network IDs. Moreover, third-party scripts including session replay libraries exfiltrate whole DOM to reproduce users' detailed interaction with the web page, while leaking personal and sensitive data including passwords, health conditions, prescriptions, credit card and student data. While the specific vulnerabilities we report have largely been fixed, the underlying root causes are harder to fix, as they stem from economic or human factors, or are baked into the security architecture of the web. We highlight the role of regulation and periodic, large-scale web privacy scanning to preventing and remedying future vulnerabilities arising from the same root causes.

10 Code and data

The source code and data from our study are publicly available at:

https://webtransparency.cs.princeton.edu/no_boundaries.

Acknowledgments

We thank our shepherd Oleksii Starov for his valuable feedback, Mihir Kshirsagar for his help with questions related to regulations, Dimitar Bounov and Sorin Lerner for bringing Gradescope leaks to our attention. This study is supported by an NSF grant (CNS 1526353). Some of our measurements were funded by an Amazon AWS Cloud Credits for Research grant. Gunes Acar holds a Postdoctoral fellowship from the Research Foundation Flanders (FWO). This work was supported by CyberSecurity Research Flanders with reference number VR20192203.

Appendix A Code snippets

Listing 2 Snippet of the `Adthink` code that inserts an invisible form to trigger browser autofill. The archived version of the script can be accessed at:https://gist.github.com/gunesacar/6de71057fc15074d94cda5c344b06cbb.

```
function ea() {
  var a = document.createElement("div");
  a.height = "1px";
  a.width = "1px";
  a.style.position = "absolute";
  a.style.top = "-42px";
  a.style.left = "-42px";
  a.style.display = "none";
  var b = document.getElementsByTagName("body
      ")[0];
  b || (b = document.documentElement);
  b.appendChild(a);
  var c = document.createElement("form");
  c.method = "POST";
  c.action = "";
  a.appendChild(c);
  var d = document.createElement("input");
  try {
    d.type = "email"
  } catch (e) {
    d.type = "text"
  }
  d.name = "email";
  d.id = "pus-email";
  d.required = !0;
  d.autocomplete = "email";
  c.appendChild(d);
  d = document.createElement("input");
  d.type = "password";
  d.name = "password";
  d.id = "password";
  d.required = !0;
  d.autocomplete = "password";
  c.appendChild(d);
  t();
  setTimeout(function() {
      b.removeChild(a)
    },
    2500)
}
```

Appendix B Bandsintown vulnerability

Some third parties use the Facebook Login feature to authenticate users across many websites: Disqus, a commenting widget, is a popular example. In a similar fashion, hidden third-party trackers can use Facebook Login to track users if they are also a first party that users visit directly. We found that this is exactly what the Bandsintown website[16] was doing. Worse, they did so in a way that allowed any malicious site to embed Bandsintown's iframe to identify its users.

At the time of measurement, Bandsintown offered an advertising service called "Amplified"[73], which we found to be present on many of the top music-related sites including lyrics.com, songlyrics.com and lyricsmania.com. When a Bandsintown user browsed to a website that embedded Bandsintown's Amplified advertising product, the advertising script would embed an invisible iframe which connected to Bandsintown's Facebook application using the authentication tokens established earlier, and would the grab the user's Facebook ID. The iframe then passed the user ID back to the embedding script.

We discovered that the iframe injected by Bandsintown would pass the user's information to the embedding script indiscriminately. Thus, any malicious site could have used their iframe to identify visitors. We informed Bandsintown of this vulnerability and they confirmed that it is now fixed [60].

Appendix C Encoding methods and hash algorithms used in leak detection

Hash algorithms Adler-32, BLAKE2b, BLAKE2s, CRC-32, MD2, MD4, MD5, MurmurHash2, MurmurHash2 (Unsigned), MurmurHash3 (32/64/128-bit), RipeMD-160, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-3(224, 256, 384, 512-bit), WHIRLPOOL.

Encodings Base16, Base32, Base58, Base64, Entity encoding, URL encoding (percent-encoding), yEnc, DEFLATE, GZIP, ZLIB (RFC 1950).

References

[1] "Facebook login," 2018. [Online]. Available: https://developers.facebook.com/docs/facebook-login/

[2] S. Englehardt and A. Narayanan, "Online tracking: A 1-million-site measurement and analysis," in *ACM Conference*

16 https://www.bandsintown.com/

on Computer and Communications Security, 2016.

[3] A. Fou. (2016) Javascript trackers open security holes | exchangewire.com. [Online]. Available: https://www.exchangewire.com/blog/2016/05/19/%E2%80%8Bon-site-javascript-trackers-open-gaping-security-holes/

[4] J. Weiler. (2016) 3rd party javascript management cheat sheet - owasp. [Online]. Available: https://www.owasp.org/index.php/3rd_Party_Javascript_Management_Cheat_Sheet#Major_risks

[5] Mozilla, "Document.referrer - Web APIs," https://developer.mozilla.org/en-US/docs/Web/API/Document/referrer, accessed: 2019-12-02.

[6] J. Eisinger and E. Stark, "Referrer Policy – W3C Candidate Recommendation," https://www.w3.org/TR/referrer-policy/, 2017, accessed: 2018-02-01.

[7] B. Krishnamurthy and C. E. Wills, "On the leakage of personally identifiable information via online social networks," in *2nd ACM workshop on Online social networks*. ACM, 2009.

[8] ——, "Privacy leakage in mobile online social networks," in *3rd conference on Online social networks*. USENIX Association, 2010.

[9] B. Krishnamurthy, K. Naryshkin, and C. Wills, "Privacy leakage vs. protection measures: the growing disconnect," in *Proceedings of W2SP*, vol. 2, 2011.

[10] J. Mayer, "Tracking the trackers: Where everybody knows your username," https://cyberlaw.stanford.edu/blog/2011/10/tracking-trackers-where-everybody-knows-your-username, 2011.

[11] 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*. ACM, 2016, pp. 361–374.

[12] A. Razaghpanah, R. Nithyanand, N. Vallina-Rodriguez, S. Sundaresan, M. Allman, C. Kreibich, and P. Gill, "Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem," in *Network and Distributed System Security Symposium (NDSS)*. IEEE, 2018.

[13] I. Reyes, P. Wijesekera, J. Reardon, A. E. B. On, A. Razaghpanah, N. Vallina-Rodriguez, and S. Egelman, ""Won't somebody think of the children?" examining COPPA compliance at scale," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 63–83, 2018.

[14] S. Jain, M. Javed, and V. Paxson, "Towards mining latent client identifiers from network traffic," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 2, pp. 100–114, 2016.

[15] O. Starov, P. Gill, and N. Nikiforakis, "Are you sure you want to contact us? quantifying the leakage of pii via website contact forms," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 1, pp. 20–33, 2016.

[16] O. Starov and N. Nikiforakis, "Extended tracking powers: Measuring the privacy diffusion enabled by browser extensions," in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 1481–1490.

[17] J. Brookman, P. Rouge, A. Alva, and C. Yeung, "Cross-device tracking: Measurement and disclosures," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 2, pp. 133–148, 2017.

[18] S. Zimmeck, J. S. Li, H. Kim, S. M. Bellovin, and T. Jebara, "A privacy analysis of cross-device tracking," in *Proceedings of the 26th USENIX Security Symposium*, 2017.

[19] D. Hedin, A. Birgisson, L. Bello, and A. Sabelfeld, "Jsflow: Tracking information flow in javascript and its apis," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 2014, pp. 1663–1671.

[20] W. De Groef, D. Devriese, N. Nikiforakis, and F. Piessens, "Flowfox: a web browser with flexible and precise information flow control," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 748–759.

[21] J. Ren, M. Lindorfer, D. J. Dubois, A. Rao, D. Choffnes, and N. Vallina-Rodriguez, "Bug fixes, improvements,... and privacy leaks," 2018.

[22] "nsILoginManager - Mozilla | MDN," May 2020, [Online; accessed 17. May 2020]. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Reference/Interface/nsILoginManager

[23] N. Vallina-Rodriguez, C. Kreibich, M. Allman, and V. Paxson, "Lumen: Fine-grained visibility and control of mobile traffic in user-space," 2017.

[24] Electronic Frontier Foundation, "EFForg/privacybadger," May 2020, [Online; accessed 16. May 2020]. [Online]. Available: https://github.com/EFForg/privacybadger

[25] G. Acar, "Instrument the stackstrace for the HTTP requests. · mozilla/OpenWPM@d659792," October 2016, [Online; accessed 17. May 2020]. [Online]. Available: https://github.com/mozilla/OpenWPM/commit/d659792766b940755634877cdc1e6a8267fa9eb7

[26] Mozilla, "Components.stack - Mozilla | MDN," 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Language_Bindings/Components.stack

[27] S. Englehardt, J. Han, and A. Narayanan, "I never signed up for this! privacy implications of email tracking," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 1, pp. 109–126, 2018.

[28] Bugzilla, "Stealing Firefox saved passwords," https://bugzilla.mozilla.org/show_bug.cgi?id=1107422.

[29] ——, "password manager + XSS = disaster," https://bugzilla.mozilla.org/show_bug.cgi?id=408531.

[30] "Princeton Web Census Data Release," 2018, [Online; accessed 9. Dec. 2019]. [Online]. Available: https://webtransparency.cs.princeton.edu/webcensus/data-release/

[31] "Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach," https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election, 2018, accessed: 2018.

[32] I. Facebook, "Zuckerberg Responses to Judiciary Committee Questions for the Record," https://www.judiciary.senate.gov/imo/media/doc/Zuckerberg%20Responses%20to%20Judiciary%20Committee%20QFRs.pdf, 2018, online; accessed 2019-02-29.

[33] "De-anonymizing Facebook's app-scoped ids," Oct 2015, [Online; accessed 11. Apr. 2020]. [Online]. Available: https://snarfed.org/2015-10-25_de-anonymizing-facebooks-app-scoped-ids

[34] "Graph API Reference v6.0: User Picture - Documentation - Facebook for Developers," Apr 2020, [Online; accessed 11.

Apr. 2020]. [Online]. Available: https://developers.facebook.com/docs/graph-api/reference/user/picture

[35] W. Palant, "palant/jsdeobfuscator: ," Dec 2017, [Online; accessed 16. May 2020]. [Online]. Available: https://github.com/palant/jsdeobfuscator

[36] "Social SDK Usage Distribution in the Top 1 Million Sites," May 2020, [Online; accessed 21. May 2020]. [Online]. Available: https://web.archive.org/web/20200507012108/https://trends.builtwith.com/javascript

[37] "beautifulsoup4," May 2020, [Online; accessed 17. May 2020]. [Online]. Available: https://pypi.org/project/beautifulsoup4

[38] "zlib — Compression compatible with gzip — Python 3.8.3 documentation," May 2020, [Online; accessed 17. May 2020]. [Online]. Available: https://docs.python.org/3/library/zlib.html

[39] FullStory, "Terms & Conditions," https://web.archive.org/web/20171115044316/https://www.fullstory.com/legal/terms-and-conditions/, 2017, accessed: 2018-05-09.

[40] SessionCam, "What information do we collect for our clients?" https://web.archive.org/web/20171115050443/https://sessioncam.com/privacy-policy-cookies/, 2017, accessed: 2018-05-09.

[41] "Show Password," May 2020, [Online; accessed 19. May 2020]. [Online]. Available: https://chrome.google.com/webstore/detail/show-password/gaichhcdflnpllpkmocfcbkbacefiank

[42] "Unmask Password," May 2020, [Online; accessed 19. May 2020]. [Online]. Available: https://chrome.google.com/webstore/detail/unmask-password/pmmeddaccflimcipblojlnfandenhicb?hl=en-US

[43] Chromium, "Users can be tracked via password manager," https://bugs.chromium.org/p/chromium/issues/detail?id=798492.

[44] Bugzilla, "Consider making signon.autofillForms = false to be the default," https://bugzilla.mozilla.org/show_bug.cgi?id=1427543.

[45] ——, "Consider imposing restrictions on tracking scripts running in the first-party context," https://bugzilla.mozilla.org/show_bug.cgi?id=1601452.

[46] Apple, "About the security content of Safari 11.1," https://support.apple.com/en-us/HT208695.

[47] A. Tseng, "Disable password autofill on page load," https://github.com/brave/browser-laptop/issues/12489, 2018, accessed: 2018-05-28.

[48] P. Snyder, B. Eich, and P. Jumde, "privacycg/js-membranes: JS Isolation via Origin Labels and Membranes," https://github.com/privacycg/js-membranes, [Online; accessed 12. Mar. 2020].

[49] MonztA, "(Comment) No boundaries: Exfiltration of personal data by session-replay scripts," https://freedom-to-tinker.com/2017/11/15/no-boundaries-exfiltration-of-personal-data-by-session-replay-scripts/#comment-28428, 2017, accessed: 2018-05-28.

[50] lukemulks, "Block ad tracking scripts used with password managers," https://github.com/disconnectme/disconnect-tracking-protection/issues/36, 2018, accessed: 2018-05-28.

[51] carbureted, "Add screen tracking services, bitcoin miners, miscellaneous third party analytics, and move wishabi.net to content," https://github.com/disconnectme/disconnect-tracking-protection/commit/09c7b279a88c8eda1ae18fe7b405e6cc315d2855, 2017, accessed: 2018-05-28.

[52] R. Brandom, "Ad targeters are pulling data from your browser's password manager," 2017. [Online]. Available: https://www.theverge.com/2017/12/30/16829804/browser-password-manager-adthink-princeton-research

[53] "Diff of the api.behavioralengine.com/scripts/be-init.js script archived by the Wayback Machine on 28 December 2017 and 3 January 2018." Jun 2020, [Online; accessed 4. Jun. 2020]. [Online]. Available: https://gist.github.com/gunesacar/11883c40b4a2def7cee0f5dd757787d6#file-onaudience_behavioral_engine-diff-L95-L134

[54] S. Englehardt, G. Acar, and A. Narayanan, "No boundaries for credentials: New password leaks to Mixpanel and Session Replay Companies," https://freedom-to-tinker.com/2018/02/26/no-boundaries-for-credentials-password-leaks-to-mixpanel-and-session-replay-companies/, 2018.

[55] "Facebook Login Changes to Address Abuse," Apr 2018, [Online; accessed 9. Dec. 2019]. [Online]. Available: https://developers.facebook.com/blog/post/2018/04/19/facebook-login-changes-address-abuse

[56] "Over 400 of the World's Most Popular Websites Record Your Every Keystroke, Princeton Researchers Find," Nov 2017, [Online; accessed 9. Dec. 2019]. [Online]. Available: https://www.vice.com/en_us/article/59yexk/princeton-study-session-replay-scripts-tracking-you

[57] N. Tiku, "The Dark Side of 'Replay Sessions' That Record Your Every Move Online," Nov 2017. [Online]. Available: https://www.wired.com/story/the-dark-side-of-replay-sessions-that-record-your-every-move-online

[58] S. Englehardt, G. Acar, and A. Narayanan, "Website operators are in the dark about privacy violations by third-party scripts," https://freedom-to-tinker.com/2018/01/12/website-operators-are-in-the-dark-about-privacy-violations-by-third-party-scripts/, 2018.

[59] G. Acar, S. Englehardt, and A. Narayanan, "No boundaries for user identities: Web trackers exploit browser login managers," https://freedom-to-tinker.com/2017/12/27/no-boundaries-for-user-identities-web-trackers-exploit-browser-login-managers/, 2017.

[60] S. Englehardt, G. Acar, and A. Narayanan, "No boundaries for Facebook data: third-party trackers abuse Facebook Login," https://freedom-to-tinker.com/2018/04/18/no-boundaries-for-facebook-data-third-party-trackers-abuse-facebook-login/, 2018.

[61] ——, "No boundaries: Exfiltration of personal data by session-replay scripts," https://freedom-to-tinker.com/2017/11/15/no-boundaries-exfiltration-of-personal-data-by-session-replay-scripts/, 2017.

[62] Apple Inc., "CVE-2018-4137," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-4137, 2018, accessed: 2018-05-28.

[63] FullStory, "Digital Experience Analytics, Session Replay, Heatmaps | FullStory," Jan 2019, [Online; accessed 28. Feb. 2020]. [Online]. Available: https://www.fullstory.com/features/session-replay

[64] M. S. Miller, M. Samuel, B. Laurie, I. Awad, and M. Stay, "Safe active content in sanitized javascript," *Google, Inc., Tech. Rep*, 2008.

[65] L. A. Meyerovich and B. Livshits, "Conscript: Specifying and enforcing fine-grained security policies for javascript in the

browser," in *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 481–496.

[66] "GDPR (General Data Protection Regulation), FullStory, and You," Jan 2020, [Online; accessed 13. Apr. 2020]. [Online]. Available: https://www.fullstory.com/resources/gdpr-and-fullstory

[67] "General Data Protection Regulation (GDPR)," Apr 2020, [Online; accessed 13. Apr. 2020]. [Online]. Available: https://www.clicktale.com/company/data-privacy/general-data-protection-regulation-gdpr

[68] "YANDEX.METRICA DATA PROCESSING AGREEMENT (DPA) - Legal documents. Help," Apr 2020, [Online; accessed 13. Apr. 2020]. [Online]. Available: https://yandex.com/legal/metrica_agreement

[69] "Add-on SDK - Archive of obsolete content | MDN," Feb 2020, [Online; accessed 26. Feb. 2020]. [Online]. Available: https://developer.mozilla.org/en-US/docs/Archive/Add-ons/Add-on_SDK

[70] Mozilla, "Restore instrumentation regarding what code is causing requests · Issue #352 · mozilla/OpenWPM," Feb 2020, [Online; accessed 26. Feb. 2020]. [Online]. Available: https://github.com/mozilla/OpenWPM/issues/352

[71] D. Zeber, S. Bird, C. Oliveira, W. Rudametkin, I. Segall, F. Wollsén, and M. Lopatka, "The representativeness of automated web crawls as a surrogate for human browsing," in *The Web Conference*, 2020.

[72] S. S. Ahmad, M. D. Dar, M. F. Zaffar, N. Vallina-Rodriguez, and R. Nithyanand, "Apophanies or epiphanies? how crawlers impact our understanding of the web," 2020.

[73] "Bandsintown Amplified," 2019, [Online; accessed 10. Dec. 2019]. [Online]. Available: https://publishers.bandsintown.com