

The Web Never Forgets: Persistent Tracking Mechanisms in the Wild

Gunes Acar¹, Christian Eubank², Steven Englehardt², Marc Juarez¹
Arvind Narayanan², Claudia Diaz¹

¹KU Leuven, ESAT/COSIC and iMinds, Leuven, Belgium
{name.surname}@esat.kuleuven.be

²Princeton University
{cge,ste,arvindn}@cs.princeton.edu

ABSTRACT

We present the first large-scale studies of three advanced web tracking mechanisms — canvas fingerprinting, evercookies and use of “cookie syncing” in conjunction with evercookies. Canvas fingerprinting, a recently developed form of browser fingerprinting, has not previously been reported in the wild; our results show that over 5% of the top 100,000 websites employ it. We then present the first automated study of evercookies and respawning and the discovery of a new evercookie vector, IndexedDB. Turning to cookie syncing, we present novel techniques for detection and analysing ID flows and we quantify the amplification of privacy-intrusive tracking practices due to cookie syncing.

Our evaluation of the defensive techniques used by privacy-aware users finds that there exist subtle pitfalls — such as failing to clear state on multiple browsers at once — in which a single lapse in judgement can shatter privacy defenses. This suggests that even sophisticated users face great difficulties in evading tracking techniques.

Categories and Subject Descriptors

K.6.m [Management of Computing and Information Systems]: Miscellaneous; H.3.5 [Information Storage and Retrieval]: Online Information Services — *Web-based services*; K.4.4 [Computers and Society]: Electronic Commerce — *Security*

Keywords

Web security; privacy; tracking; canvas fingerprinting; browser fingerprinting; cookie syncing; evercookie, JavaScript; Flash

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CCS'14, November 3–7, 2014, Scottsdale, Arizona, USA.

Copyright is held by the owner/author(s). Publication rights licensed to ACM.

ACM 978-1-4503-2957-6/14/11 ...\$15.00.

<http://dx.doi.org/10.1145/2660267.2660347>.

1. INTRODUCTION

A 1999 New York Times article called cookies comprehensive privacy invaders and described them as “surveillance files that many marketers implant in the personal computers of people.” Ten years later, the stealth and sophistication of tracking techniques had advanced to the point that Edward Felten wrote “If You’re Going to Track Me, Please Use Cookies” [18]. Indeed, online tracking has often been described as an “arms race” [47], and in this work we study the latest advances in that race.

The tracking mechanisms we study are advanced in that they are hard to control, hard to detect and resilient to blocking or removing. *Canvas fingerprinting* uses the browser’s Canvas API to draw invisible images and extract a persistent, long-term fingerprint without the user’s knowledge. There doesn’t appear to be a way to automatically block canvas fingerprinting without false positives that block legitimate functionality; even a partial fix requires a browser source-code patch [40]. *Evercookies* actively circumvent users’ deliberate attempts to start with a fresh profile by abusing different browser storage mechanisms to restore removed cookies. *Cookie syncing*, a workaround to the Same-Origin Policy, allows different trackers to share user identifiers with each other. Besides being hard to detect, cookie syncing enables back-end server-to-server data merges hidden from public view.

Our goal is to improve transparency of web tracking in general and advanced tracking techniques in particular. We hope that our techniques and results will lead to better defenses, increased accountability for companies deploying exotic tracking techniques and an invigorated and informed public and regulatory debate on increasingly persistent tracking techniques.

While conducting our measurements, we aimed to automate all possible data collection and analysis steps. This improved the scalability of our crawlers and allowed us to analyze 100,000 sites for fingerprinting experiments, as well as significantly improve upon the scale and sophistication of the prior work on respawning, evercookies and cookie syncing.

1.1 Contributions

First study of real-world canvas fingerprinting practices. We present the results of previously unreported

canvas fingerprinting scripts as found on the top 100,000 Alexa sites. We find canvas fingerprinting to be the most common fingerprinting method ever studied, with more than 5% prevalence. Analysis of the real-world scripts revealed that they went beyond the techniques suggested by the academic research community (Section 3).

Automated analysis of evercookies and respawning. We describe an automated detection method for evercookies and cookie respawning. Applying this analysis, we detected respawning by Flash cookies on 10 of the 200 most popular sites and found 33 different Flash cookies were used to respawn over 175 HTTP cookies on 107 of the top 10,000 sites. We also uncover a new evercookie vector, IndexedDB, which was never found in the wild before (Section 4). Remarkably, respawning has already led to a lawsuit and a \$500,000 settlement [14], and yet it is quite prevalent on the web.

Cookie syncing privacy analysis. We find instances of syncing of respawned IDs in the wild, i.e., an ID respawned by one domain is passed to another domain. Respawning enables trackers to link a user’s browsing logs before cookie clearing to browsing logs after cookie clearing. In our measurements, approximately 1.4% of a user’s browser history can be linked this way in the wild. However, the figure jumps to at least 11% when these respawned cookies are subsequently synced. Cookie syncing also allows trackers to merge records on individual users, although this merging cannot be observed via the browser. Our measurements in Section 5 show that in the model of back-end merging we study, the number of trackers that can obtain a sizable fraction (40%) of a user’s browsing history increases from 0.3% to 22.1%.

Novel techniques. In performing the above experiments, we developed and utilized novel analysis and data collection techniques that can be used in similar web privacy studies.

- Using the *strace* debugging tool for low-level monitoring of the browser and the Flash plugin player (Section 4.2).
- A set of criteria for distinguishing and extracting pseudonymous identifiers from traditional storage vectors, such as cookies, as well as other vectors such as Flash storage. By extracting known IDs, we can track them as they spread to multiple domains through cookie syncing.

Making the code and the data public. We intend to publicly release all the code we developed for our experiments and all collected data, including (i) our crawling infrastructure, (ii) modules for analysing browser profile data and (iii) crawl databases collected in the course of this study.

1.2 Implications

The thrust of our results is that the three advanced tracking mechanisms we studied are present in the wild and some of them are rather prevalent. As we elaborate on in Section 6.1, they are hard to block, especially without loss of content or functionality, and once some tracking has happened, it is hard to start from a truly clean profile. A frequent argument in online privacy debates is that individuals should “take control” of their own privacy online. Our results suggest that even sophisticated users may not be able to do so without significant trade-offs.

We show that cookie syncing can greatly amplify privacy breaches through server-to-server communication. While web privacy measurement has helped illuminate many privacy breaches online, server-to-server communication is not directly observable. All of this argues that greater oversight over online tracking is becoming ever more necessary.

Our results only apply to desktop browsing. Studying similar tracking mechanisms on mobile platforms requires distinct methodologies and infrastructure and is left to future work.

2. BACKGROUND AND RELATED WORK

The tracking mechanisms studied in this paper can be differentiated from their conventional counterparts by their potential to circumvent users’ tracking preferences, being hard to discover and resilient to removal. We selected three of the most prominent persistent tracking techniques — canvas fingerprinting, evercookies and cookie syncing — based on the lack of adequate or comprehensive empirical measurements of these mechanisms in the wild. We now give a brief overview of these techniques.

Canvas fingerprinting: Canvas fingerprinting is a type of browser or device fingerprinting technique that was first presented in a paper by Mowery and Shacham in 2012 [32]. The authors found that by using the Canvas API of modern browsers, an adversary can exploit subtle differences in the rendering of the same text or WebGL scenes to extract a consistent fingerprint that can easily be obtained in a fraction of a second without user’s awareness.

The same text can be rendered in different ways on different computers depending on the operating system, font library, graphics card, graphics driver and the browser. This may be due to the differences in font rasterization such as anti-aliasing, hinting or sub-pixel smoothing, differences in system fonts, API implementations or even the physical display [32]. In order to maximize the diversity of outcomes, the adversary may draw as many different letters as possible to the canvas. Mowery and Shacham, for instance, used the pangram *How quickly daft jumping zebras vex* in their experiments.

The entropy available in canvas fingerprints has never been measured in a large-scale published study like Panopticlick [16]. Mowery and Shacham collected canvas fingerprints from 294 Mechanical Turk users and computed 5.73 bits of entropy for their dataset. Since this experiment was significantly limited for measuring the canvas fingerprinting entropy, they had a further estimate of at least 10 bits, meaning one in a thousand users share the same fingerprint [32].

Figure 1 shows the basic flow of operations to fingerprint canvas. When a user visits a page, the fingerprinting script first draws text with the font and size of its choice and adds background colors (1). Next, the script calls Canvas API’s `ToDataURL` method to get the canvas pixel data in *dataURL* format (2), which is basically a Base64 encoded representation of the binary pixel data. Finally, the script takes the hash of the text-encoded pixel data (3), which serves as the fingerprint and may be combined with other high-entropy browser properties such as the list of plugins, the list of fonts, or the user agent string [16].

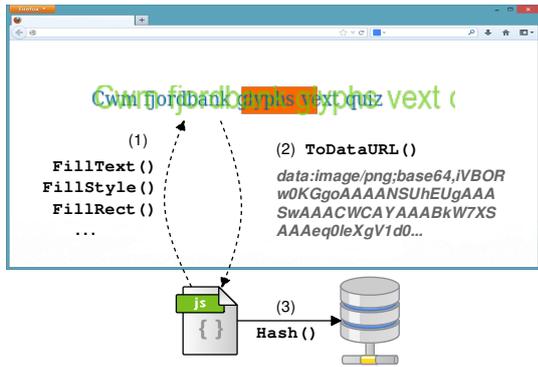


Figure 1: Canvas fingerprinting basic flow of operations

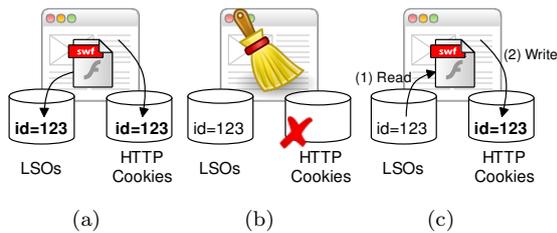


Figure 2: Respawning HTTP cookies by Flash evercookies: (a) the webpage stores an HTTP and a Flash cookie (LSO), (b) the user removes the HTTP cookie, (c) the webpage respawns the HTTP cookie by copying the value from the Flash cookie.

Evercookies and respawning: A 2009 study by Soltani et al. showed the abuse of Flash cookies for regenerating previously removed HTTP cookies, a technique referred to as “respawning” [43]. They found that 54 of the 100 most popular sites (rated by Quantcast) stored Flash cookies, of which 41 had matching content with regular cookies. Soltani et al. then analyzed respawning and found that several sites, including aol.com, about.com and hulu.com, regenerated previously removed HTTP cookies using Flash cookies. A follow up study in 2011 found that sites use ETags and HTML5 localStorage API to respawn cookies [7].

In 2010, Samy Kamkar demonstrated the “Evercookie,” a resilient tracking mechanism that utilizes multiple storage vectors including Flash cookies, localStorage, sessionStorage and ETags [21]. Kamkar employed a variety of novel techniques, such as printing ID strings into a canvas image which is then force-cached and read from the cached image on subsequent visits. Instead of just respawning HTTP cookies by Flash cookies, his script would check the cleared vectors in the background and respawn from any storage that persists.

Figure 2 depicts the stages of respawning by Local Shared Objects (LSOs), also known as Flash cookies. Whenever a user visits a site that uses evercookies, the site issues an ID and stores it in multiple storage mechanisms, including cookies, LSOs and localStorage. In Figure 2a, the value 123 is stored in both HTTP and Flash cookies. When the user removes her HTTP cookie (Figure 2b), the website places a cookie with the same value (123) by reading the ID value

from a Flash cookie that the user may fail to remove (Figure 2c).

Cookie syncing: Cookie synchronization or cookie syncing is the practice of tracker domains passing pseudonymous IDs associated with a given user, typically stored in cookies, amongst each other. Domain A, for instance, could pass an ID to domain B by making a request to a URL hosted by domain B which contains the ID as a parameter string. According to Google’s developer guide to cookie syncing (which they call cookie matching), cookie syncing provides a means for domains sharing cookie values, given the restriction that sites can’t read each other cookies, in order to better facilitate targeting and real-time bidding [4].

In general, we consider the domains involved in cookie syncing to be third parties — that is, they appear on the first-party sites that a user explicitly chooses to visit. Although some sites such as facebook.com appear both in a first and third-party context, this distinction is usually quite clear.

The authors of [38] consider cookie synchronization both as a means of detecting business relationships between different third-parties but also as a means of determining to what degree user data may flow between parties, primarily through real-time bidding. In the present work, we study the implications of the fact that trackers that share an ID through syncing are in position to merge their database entries corresponding to a particular user, thereby reconstructing a larger fraction of the user’s browsing patterns.

2.1 Related work

While HTTP cookies continue to be the most common method of third-party online tracking [41], a variety of more intrusive tracking mechanisms have been demonstrated, refined and deployed over the last few years. In response, various defenses have been developed, and a number of studies have presented measurements of the state of tracking. While advertising companies have claimed that tracking is essential for the web economy to function [42], a line of research papers have proposed and prototyped solutions to carry out behavioral advertising without tracking.

Fingerprinting, novel mechanisms. Researchers have presented novel browser fingerprinting mechanisms such as those based on performance metrics [31], the JavaScript engine [33], the rendering engine [50], clock skew [23], WebGL and canvas fingerprinting [32]. Most of those studies followed the path opened by the influential Panoptick study [16], which demonstrated the potentials of browser fingerprinting for online tracking.

Measurement studies. Web privacy measurement is a burgeoning field; an influential early work is [25] and prominent recent work includes [29, 41]. Mayer and Mitchell made a comprehensive survey of tracking in combination with the policy that surrounds it, and developed a tool for similar web privacy measurement studies [29]. Roesner et al. analyzed different tracking methods and suggested a taxonomy for third-party tracking [41].

Other papers have looked at various aspects of web privacy, including PII leakage [26], mobile web tracking [17], JavaScript inclusions [35], targeted advertisements [27], and the effectiveness of blocking tools [28].

Two studies measured the prevalence of different fingerprinting mechanisms and evaluated existing countermeasures [37, 6]. Nikiforakis et al. studied three previ-

ously known fingerprinting companies and found 40 such sites among the top 10K sites employing practices such as font probing and the use of Flash to circumvent proxy servers [37]. Acar et al. found that 404 sites in the top million deployed JavaScript-based fingerprinting and 145 sites of the top 10,000 sites leveraged Flash-based fingerprinting [6].

In comparison to these studies, we focus on canvas fingerprinting, which, to the best of our knowledge, has never been reported to be found in the wild and is much harder to block.

Several studies have looked at the use of Flash cookies (LSOs) and, in particular, the use of Flash cookies to respawn HTTP cookies [43, 7, 30]. Soltani et al. uncovered the first use of respawning by Flash cookies [43], and in a follow-up study, Ayenson et al. found the first use of cache ETags and localStorage for respawning [7]. McDonald and Cranor analyzed the landing pages of 100 popular websites, plus 500 randomly-selected websites, and found two cases of respawning in the top 100 websites and no respawning in the randomly selected 500 sites [30]. In a recent study, Sorensen analyzed the use of cache as a persistent storage mechanism and found several instances of HTTP cookies respawned from cached page content [44]. The main difference between our study and the papers mentioned here is that we automated respawning detection as explained in Section 4, and this allowed us to analyze orders of magnitude more sites.

Olejnik et al. studied *cookie syncing* (which they call cookie matching) [38]. They found that over 100 cookie syncing events happen on the top 100 sites. In comparison to their work, our study of cookie syncing (i) is large-scale, covering 3,000 sites, (ii) is based on crawling rather than crowd-sourcing, allowing easier comparative measurements over time and (iii) presents a global view, in that we go beyond detecting individual sync events and are able to capture and analyze the propagation of IDs through the tracking ecosystem. Further, we study how cookie syncing interacts with respawning, leading to more persistent tracking and widening the effects of these two vulnerabilities taken individually.

Program analysis of JavaScript (i.e., static analysis and dynamic analysis) is a common technique in web security [46]. A few studies have used such techniques for blocking or measuring web trackers. Orr et al. use static analysis to detect and block JavaScript-loaded ads [39]. Tran et al. use dynamic taint analysis to detect various privacy-invasive behaviors [48]. Acar et al. use behavioral analysis to detect fingerprinting scripts that employ font probing [6].

Defenses. Besson et al. [10] examined the theoretical boundaries of fingerprinting defenses using Quantified Information Flow. Following a more practical approach, Nikiforakis and others developed a defense called PriVaricator to prevent linkability from fingerprinters by randomizing browser features such as plugins [36]. Finally, Unger et al. [50], studied the potentials of browser fingerprinting as a defense mechanism against HTTP(S) session hijacking.

In Section 6.1 we discuss how existing privacy tools defend against the advanced tracking mechanisms we study.

Behavioral targeting without tracking. Several papers have addressed the question of whether all this tracking is in fact necessary — they proposed ways to achieve the purported goals of third-party tracking, primarily targeted

advertising, without server-side profiles. In *Adnostic*, the browser continually updates a behavioral profile of the user based on browsing activity, and targeting is done locally [14]. *PrivAd* has a similar model, but includes a trusted party that attempts to anonymize the client [20]. *RePriv* has the more general goal of enabling personalization via interest profiling in the browser [19]. Bilenko et al. propose a model in which the user’s profile and recent browsing history is stored in a cookie [11]. Other work on similar lines includes [8, 49, 34].

3. CANVAS FINGERPRINTING

Canvas fingerprinting works by drawing text onto canvas and reading the rendered image data back. In the following experiments we used an instrumented Firefox browser that we built by modifying the source code and logged all the function calls that might be used for canvas fingerprinting.

3.1 Methodology and Data collection

Our methodology can be divided into two main steps. In the first, we identified the ways we can detect canvas fingerprinting, developed a crawler based on an instrumented browser and ran exploratory crawls. This stage allowed us to develop a formal and automated method based on the early findings. In the second step, we applied the analysis method we distilled from the early findings and nearly fully automated the detection of canvas fingerprinting.

Mowery and Shacham used `fillText` and `ToDataURL` methods to draw text and read image data respectively [32]. We logged the return value of `ToDataURL` and, in order to find out the strings drawn onto the canvas, we logged the arguments of `fillText` and `strokeText` methods¹.

We logged the URL of the caller script and the line number of the calling (initiator) code using Firefox’s `nsContentUtils::GetCurrentJSContext` and `nsJSUtils::GetCallingLocation` methods. This allowed us to precisely attribute the fingerprinting attempt to the responsible script and the code segment. All function call logs were parsed and combined in a SQLite database that allowed us to efficiently analyze the crawl data. For each visit, we also added cookies, localStorage items, cache metadata, HTTP request/response headers and request bodies to the SQLite database. We used mitmproxy² to capture HTTP data and parsed data accumulated in the profile folder for other data such as cookies, localStorage and cache data. The aggregated data were used in the early stage analysis for canvas fingerprinting and evercookie detection, which is explained in Section 4.2. Our browser modifications for Firefox consist of mere 33 lines of code, spread across four files and the performance overhead of the modifications is minimal.

We crawled the home pages of the top 100,000 Alexa sites with the instrumented Firefox browser between 1-5 May 2014. We used Selenium [5] to drive browsers to sites and ran multiple Firefox instances in parallel to reduce the

¹In addition to these three methods we intercepted calls to `MozFetchAsStream`, `getImageData` and `ExtractData` methods which can be used to extract canvas image data. But we did not put effort into recording the extracted image data for three reasons: they were not used in the original canvas fingerprinting paper [32], they are less convenient for fingerprinting (requires extra steps), and we did not find any script that uses these methods and fingerprints other browser properties in the initial experiments.

²<http://mitmproxy.org/>

crawl time. Implementing some basic optimizations and a naive load limiting check, we were able to run up to 30 browsers in parallel on a 4-core 8GB desktop machine running GNU/Linux operating system. The modified browsers were run in a *chroot jail* to limit the effects of the host operating system.

False positive removal The Canvas API is used by many benign scripts to draw images, create animations or store content for games. During our crawls we found interesting use cases, such as generating dynamic *favicons*, creating tag clouds, and checking font smoothing support. By examining the distinctive features of false positives and the fingerprinting scripts found in the initial experiments, we distilled the following conditions for filtering out false positives:

- There should be both `ToDataURL` and `fillText` (or `strokeText`) method calls and both calls should come from the same URL.
- The canvas image(s) read by the script should contain more than one color and its(their) aggregate size should be greater than *16x16 pixels*.
- The image should *not* be requested in a *lossy compression format* such as JPEG.

Checking the origin of the script for both read and write access helped us to remove scripts that use canvas for only generating images but not reading them or vice versa. Although it is possible that two scripts from the same domain can divide the work to circumvent our detection method, we accepted that as a limitation.

Enforcing a 16x16 pixel size limit allowed us to filter out scripts that read too few pixels to efficiently extract the canvas fingerprint. Although there are $2^{8 \times 16}$ possible color combinations for a 16x16 pixel image³, operating systems or font libraries only apply anti-aliasing (which is an important source of diversity for canvas fingerprinting) to text larger than a minimum font size.⁴

The final check was to filter out cases where canvas image data is requested in a lossy compression format. Under a lossy compression scheme, the returned image may lose the subtle differences that are essential for fingerprinting.

Applying these checks, we reduced the false positive ratio to zero for the 100,000 crawl, upon which we perform our primary analysis. We used static analysis to make sure the scripts we flagged as canvas fingerprinting were also collecting other high-entropy browser properties such as plugins, navigator features and screen dimensions. It should be noted that in other pilot crawls (beyond 100K), we witnessed some false positives that our conditions failed to remove. Also, we believe that a determined tracker may potentially circumvent our detection steps using more advanced but less reliable attacks such as pixel stealing using SVG filters [45] or CSS shaders [24].

³ $2^{\text{colordepth}^{w \times h}}$, $2^{32^{16 \times 16}} = 2^{8192}$ for the RGBA color space, which uses 24 bits for the colors (RGB) and 8 bits for the alpha channel. See, <http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#pixel-manipulation>

⁴https://wiki.ubuntu.com/Fonts#Font_Smoothing

3.2 Results

Table 1 shows the prevalence of the canvas fingerprinting scripts found during the home page crawl of the Top Alexa 100,000 sites. We found that more than 5.5% of crawled sites actively ran canvas fingerprinting scripts on their home pages. Although the overwhelming majority (95%) of the scripts belong to a single provider (*addthis.com*), we discovered a total of 20 canvas fingerprinting provider domains, active on 5542 of the top 100,000 sites⁵. Of these, 11 provider domains, encompassing 5532 sites, are third parties. Based on these providers' websites, they appear to be companies that deploy fingerprinting as part of some other service rather than offering fingerprinting directly as a service to first parties. We found that the other nine provider domains (active on 10 sites) are in-house fingerprinting scripts deployed by first parties. Note that our crawl in this paper was limited to home pages. A deeper crawl covering internal pages of the crawled sites could find a higher percentage of fingerprinting.

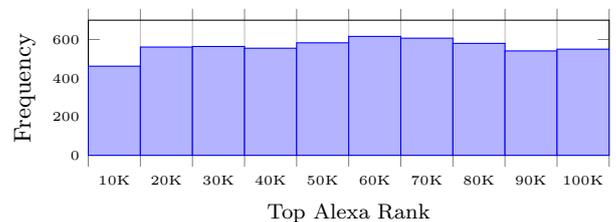


Figure 3: Frequency of canvas fingerprinting scripts on the home pages of Top Alexa 100K sites.

The 5.5% prevalence is much higher than what other fingerprinting measurement studies had previously found (0.4% [37], 0.4%, 1.5% [6]), although these studies may not be directly comparable due to the differences in methodology and data collection. Also note that canvas fingerprinting was first used by AddThis between January 15 to February 1st, 2014,⁶ which was after all the mentioned studies.

| Rank interval | % of sites with canvas fingerprinting scripts |
|---------------|---|
| [1, 1K) | 1.80 |
| [1K, 10K) | 4.93 |
| [10K, 100K] | 5.73 |

Table 2: Percentage of sites that include canvas fingerprinting scripts on the homepage, found in top 100K Alexa sites divided in intervals of variable length. Websites in the 1 to 1K rank interval are 2.5 times less likely to embed a canvas fingerprinting script than a site within 1K-10K interval.

Below rank 10,000, the prevalence of canvas fingerprinting is close to uniform. However, we found that the top 1,000 sites are 2.5 times less likely to have included canvas

⁵We discarded some cases where the canvas fingerprinting script is served from a content delivery network (CDN) and additional analysis was needed to distinguish between different providers serving from the same (CDN) domain. Including these cases would only change the number of unique sites with canvas fingerprinting to 5552 (from 5542).

⁶The date was determined using <http://httparchive.org/>

| Fingerprinting script | Number of including sites | Text drawn into the canvas |
|--|-------------------------------------|---|
| ct1.addthis.com/static/r07/core130.js | 5282 | Cwm fjordbank glyphs vext quiz, ☺ |
| i.ligatus.com/script/fingerprint.min.js | 115 | http://valve.github.io |
| src.kitcode.net/fp2.js | 68 | http://valve.github.io |
| admicro1.vcmedia.vn/fingerprint/figp.js | 31 | http://admicro.vn/ |
| amazonaws.com/af-bdaz/bquery.js | 26 | Centillion |
| *.shorte.st/js/packed/smeadvert-intermediate-ad.js | 14 | http://valve.github.io |
| stat.ringier.cz/js/fingerprint.min.js | 4 | http://valve.github.io |
| cya2.net/js/STAT/89946.js | 3 | ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz0123456789+/- |
| images.revtrax.com/RevTrax/js/fp/fp.min.jsp | 3 | http://valve.github.io |
| pof.com | 2 | http://www.plentyoffish.com |
| *.rackcdn.com/mongoose.fp.js | 2 | http://api.gonorthleads.com |
| 9 others* | 9 | (Various) |
| TOTAL | 5559 (5542 unique ¹) | - |

Table 1: Canvas fingerprinting domains found on Top Alexa 100K sites.

*: Some URLs are truncated or omitted for brevity. See Appendix for the complete list of URLs.

1: Some sites include canvas fingerprinting scripts from more than one domain.

fingerprinting scripts than the ones within the 1,000-10,000 range.

Note that the URL <http://valve.github.io>, printed by many scripts onto the canvas, belongs to the developer of an open source fingerprinting library⁷. Furthermore, all scripts except one use the same colors for the text and background shape. This similarity is possibly due to the use of the publicly available open source fingerprinting library *fingerprintjs* [51]. Figure 4 shows five different canvas images used by different canvas fingerprinting scripts. The images are generated by intercepting the canvas pixel data extracted by the scripts listed in Table 1.



Figure 4: Different images printed to canvas by fingerprinting scripts. Note that the phrase “Cwm fjordbank glyphs vext quiz” in the top image is a *perfect pangram*, that is, it contains all the letters of the English alphabet only once to maximize diversity of the outcomes with the shortest possible string.

Manually analyzing AddThis’s script, we found that it goes beyond the ideas previously discussed by researchers

⁷See, <https://github.com/Valve/fingerprintjs/blob/v0.5.3/fingerprint.js#L250>

and adds new tests to extract more entropy from the canvas image. Specifically, we found that in addition to the techniques outlined in Mowery and Shacham’s canvas fingerprinting paper [32] AddThis scripts perform the following tests:

- Drawing the text twice with different colors and the default fallback font by using a fake font name, starting with “no-real-font-”.
- Using the perfect pangram⁸ “Cwm fjordbank glyphs vext quiz” as the text string
- Checking support for drawing Unicode by printing the character $U+1F603$ a smiling face with an open mouth.
- Checking for canvas *globalCompositeOperation* support.
- Drawing two rectangles and checking if a specific point is in the path by the *isPointInPath* method.

By requesting a non-existent font, the first test tries to employ the browser’s default fallback font. This may be used to distinguish between different browsers and operating systems. Using a perfect pangram, which includes a single instance of each letter of the English alphabet, the script enumerates all the possible letter forms using the shortest string. The last three tests may be trying to uncover the browser’s support for the canvas features that are not equally supported. For instance, we found that the Opera browser cannot draw the requested Unicode character, $U+1F603$.

Another interesting canvas fingerprinting sample was the script served from the *admicro.vcmedia.vn* domain. By inspecting the source code, we found that the script checks the existence of 1126 fonts using JavaScript font probing.

⁸http://en.wikipedia.org/wiki/List_of_pangrams#Perfect_pangrams_in_English_.2826_letters.29

Overall, it is interesting to see that commercial tracking companies are advancing the fingerprinting technology beyond the privacy/security literature. By collecting fingerprints from millions of users and correlating this with cookie based identification, the popular third party trackers such as AddThis are in the best position to both measure how identifying browser features are and develop methods for monitoring and matching changing fingerprints. Note that according to a recent ComScore report, AddThis “solutions” reaches 97.2% of the total Internet population in the United States and get 103 billion monthly page views.⁹

4. EVERCOOKIES

Evercookies are designed to overcome the “shortcomings” of the traditional tracking mechanisms. By utilizing multiple storage vectors that are less transparent to users and may be more difficult to clear, evercookies provide an extremely resilient tracking mechanism, and have been found to be used by many popular sites to circumvent deliberate user actions [43, 7, 14]. In this section, we first provide a set of criteria that we used to automatically detect identifier strings, present detailed results of an automated analysis of respawning by Flash evercookies, and show the existence of respawning by both HTTP cookies and IndexedDB.

4.1 Detecting User IDs

Given that not all instances of the various potential storage vectors are used to track users, detecting evercookies hinges on determining whether a given string can serve as a user ID. In order to detect persistent IDs in a given storage vector, we leveraged data from two simultaneous crawls on separate machines and applied the following set rule set for determining which elements are identifying. We present the rules with respect to HTTP cookies but note that they are applicable to other storage locations of a similar format.

- Eliminate cookies that expire within a month of being placed. These are too transient to track a user over time.
- Parse cookie value strings using common delimiters (e.g. `:` and `&`). This extracts potentially identifying strings from non-essential data.
- Eliminate parsed fields which don’t remain constant throughout an individual crawl. Identifiers are likely to be unchanging.
- Compare instances of matching parsed cookie fields (for cookies with the same domain and name) between two unrelated crawls on different machines.
 - Eliminate fields which are not the same length.
 - Eliminate fields which are more than 33% similar according to the Ratcliff-Obershelp algorithm [12]. These are unlikely to contain sufficient entropy.

The presented method provides a strict and conservative detection of identifiers that we believe (through manual inspection) to have a very low false positive rate. We anticipate several sources of false negatives, for example ID strings

⁹<http://www.businesswire.com/news/home/20131113005901/en/comScore-Ranks-AddThis-1-Distributed-Content-United>

that are obfuscated or embedded in longer strings using non-standard delimiters or ID strings that happen to have a high similarity. Similarly, an adversarial tracker could continually change an identifier or cookie sync short-lived identifiers, but keep a mapping on the back end to enable long-term tracking. Therefore, the results of this analysis provide a lower bound on the presence of evercookie storage vectors and on the level of cookie syncing.

4.2 Flash cookies respawning HTTP cookies

Although there are many “exotic” storage vectors that can be used to store tracking identifiers, Flash cookies have a clear advantage of being shared between different browsers that make use of the Adobe Flash plugin¹⁰. We developed a procedure to automate the detection of respawning by Flash cookies employing the method discussed in Section 4.1 to detect IDs and using GNU/Linux’s *strace* [22] debugging tool to log access to Flash cookies.

Compared to earlier respawning studies [43, 7, 30], the method employed in this paper is different in terms of automation and scale. In prior studies, most of the work, including the matching of HTTP and Flash cookie identifiers was carried out manually. By automating the analysis and parallelizing the crawls, we were able to analyze 10,000 websites, which is substantially more than the previous studies (100 sites, 700 sites). Note that, similar to [30], we only visited the home pages, whereas [43, 7] visited 10 internal links on each website. Another methodological difference is that we maintained the Flash cookies when visiting different websites, whereas [43, 7] used a virtual machine to prevent contamination. Last, [30] also used the moving and contrasting Flash cookies from different computers to determine ID and non-ID strings, which is one of the main ideas of the analysis described below.

For this analysis we used data from four different crawls. First, we sequentially crawled the Alexa top 10,000 sites and saved the accumulated HTTP and Flash cookies (*Crawl₁*). We then made three 10,000 site crawls, two of which were run with the Flash cookies loaded from the sequential crawl (*Crawl_{2,3}*). The third crawler ran on a different machine, without any data loaded from the previous crawl (*Crawl₄*). Note that, except for the sequential crawl (*Crawl₁*), we ran multiple browsers in parallel to extend the reach of the study at the cost of not keeping a profile state (cookies, *localStorage*) between visits. During each visit, we ran an *strace* instance that logs all open, read and write system calls of Firefox and all of its child processes. Trace logs were parsed to get a list of Flash cookies accessed during the visit, which are then parsed and inserted into a crawl database.

For the analysis, we first split the Flash cookie contents from the three crawls (*Crawl_{2,3,4}*) by using a common set of separators (e.g. `"=:&:`). We then took the common strings between crawls made with the same LSOs (*Crawl_{2,3}*) and subtracted the strings found in LSO contents from the unrelated crawl (*Crawl₄*). We then checked the cookie contents from the original profile (*Crawl₁*) and cookies collected during the visits made with the same LSO set (*Crawl_{2,3}*). Finally, we subtracted strings that are found in an unrelated visit’s cookies (*Crawl₄*) to minimize the false positives. Note that, in order to further eliminate false positives, one can use cookies and LSOs from other unrelated crawls since an ID-

¹⁰iOS based devices and Chrome/Chromium bundled with the Pepper API are exceptions

string cannot be present in unrelated crawls. We used the 100K crawl described in the canvas fingerprinting experiments for this purpose.

For clarity, we express a simplified form of the operation in set notation:

$$\text{MaxRank} \bigcup_{i=1} \left(((F_{2_i} \cap F_{3_i}) \setminus F_4) \cap C_{2_i} \cap C_{3_i} \setminus C_4 \right),$$

where F_{n_i} denotes *Flash cookies* from $Crawl_n$ for the site with the Alexa rank i , C_{n_i} denotes *Cookies* from $Crawl_n$ for the site with the Alexa rank i and F_4 , and C_4 denotes all Flash cookies and HTTP cookies collected during $Crawl_4$.

We applied the method described above to four crawls run in May 2014 and found that 33 different Flash cookies from 30 different domains respawned a total of 355 cookies on 107 first party domains during the two crawls ($Crawl_{2,3}$). Table 3 shows that on six of the top 100 sites, Flash cookies are used to respawn HTTP cookies. Nine of top ten sites on which we observed respawning belong to Chinese companies (one from Hong Kong) whereas the other site belongs to the top Russian search engine Yandex. The Flash cookie that respawned the most cookies (69 cookies on 24 websites) was *bbcookie.sol* from the *bbcdn-bbnaut.ibillboard.com* domain which belongs to a company that is found to use Flash based fingerprinting [6]. Note that this Flash cookie respawned almost three HTTP cookies per site which belong to different third party domains (*bbelements.com*, *.ibillboard.com* and the first-party domain). The domain with the second highest number of respawns was *kiks.yandex.ru* which restored 11 cookies on 11 sites in each crawl ($Crawl_{2,3}$).

| Global rank | Site | CC | Respawning (Flash) domain | 1st/3rd Party |
|-------------|-------------|----|---------------------------|---------------|
| 16 | sina.com.cn | CN | simg.sinajs.cn | 3rd* |
| 17 | yandex.ru | RU | kiks.yandex.ru | 1st |
| 27 | weibo.com | CN | simg.sinajs.cn | 3rd* |
| 41 | hao123.com | CN | ar.hao123.com | 1st |
| 52 | sohu.com | CN | tv.sohu.com | 1st |
| 64 | ifeng.com | HK | y3.ifengimg.com | 3rd* |
| 69 | youku.com | CN | irs01.net | 3rd |
| 178 | 56.com | CN | irs01.net | 3rd |
| 196 | letv.com | CN | irs01.net | 3rd |
| 197 | tudou.com | CN | irs01.net | 3rd |

Table 3: Top-ranked websites found to include respawning based on Flash cookies. CC: ISO 3166-1 code of the country where the website is based. 3rd*: The domains that are different from the first-party but registered for the same company in the WHOIS database.

IndexedDB as Evercookie While running crawls for canvas fingerprinting experiments, we looked for sites that store data in the IndexedDB storage vector. Specifically, we checked the *storage/persistent* directory of the Firefox profile. A very small number of sites, only 20 out of 100K, were found to use the IndexedDB storage vector. Analyzing the IndexedDB file from the respawning crawl ($Crawl_2$) described above, we found that a script from the *weibo.com* domain stored an item in the IndexedDB that

exactly matched the content of the Flash cookie named *simg.sinajs.cn/stonecc_superusercookie.sol*. This Flash cookie was used to respawn HTTP cookies on Chinese microblogging site *weibo.com* and its associated web portal *sina.com.cn*. To the best of our knowledge, this is the first report of IndexedDB as an evercookie vector. A more thorough study of respawning based on IndexedDB is left for future study.

4.3 HTTP cookies respawning Flash cookies

We ran a sequential crawl of the Top 3,000 Alexa sites and saved the accumulated HTTP and Flash cookies. We extracted IDs from this crawl’s HTTP cookies using the method described in Section 4.1. We then made an additional sequential crawl of the Top 3,000 Alexa sites on a separate machine loading only the HTTP cookies from the initial crawl.

Our method of detecting HTTP respawning from Flash cookies is as follows: (i) take the intersection of the initial crawl’s flash objects with the final crawl’s flash objects (ii) subtract common strings from the intersection using an unrelated crawl’s flash objects and (iii) search the resulting strings for the first crawl’s extracted HTTP cookie IDs as described in Section 4.1. This enables us to ensure that the IDs are indeed found in the Flash objects of both crawls, aren’t common to unrelated crawls, and exist as IDs on the original machine. Using this method, we detected 11 different unique IDs common between the three storage locations.

These 11 IDs correspond to 14 first-party domains, a summary of which is provided by Table 8 in the Appendix. We primarily observe respawning from JavaScript originating from two third-parties: *www.iovation.com*, a fraud detection company that is specialized in device fingerprinting, and *www.postaffiliatepro.com*, creators of affiliate tracking software (that runs in the first-party context). We also observe three instances of what appears to be in-house respawning scripts from three brands: Twitch Interactive (*twitch.tv* and *justin.tv*), *casino.com*, and *xlovecam.com*.

5. COOKIE SYNCING

Cookie synchronization — the practice of third-party domains sharing pseudonymous user IDs typically stored in cookies — provides the potential for more effective tracking, especially when coupled with technologies such as evercookies. First, pairs of domains who both know the same IDs via synchronization can use these IDs to merge their tracking databases on the back end. Second, respawned cookies may contain IDs that are widely shared due to prior sync events, enabling trackers to link a user’s browsing histories from before and after clearing browsing state.

In this section, we present our method for detecting syncs, present an overview of the synchronization landscape and examine the threats of back-end database merges and history-linking for users who clear state.

5.1 Detecting cookie synchronization

Using the techniques outlined in Section 4.1, we identified cookies containing values likely to be user IDs. In order to learn which domains know a given ID through synchronization, we examined cookie value strings and HTTP traffic.

If a domain owns a cookie containing an ID, clearly the domain knows that ID. In fact, a telltale sign of cookie syncing is multiple domains owning cookies containing the same ID. Likewise, if an ID appears anywhere in a domain’s URL string (e.g. in the URL parameters), then that domain also knows the ID. Note that a given tracker may simply ignore an ID received during a sync, but as we will demonstrate in Section 5.3, trackers opting to store IDs have the ability to gain user data through history merging.

The domains involved in HTTP traffic can be divided into (*referrer, requested URL, location*) tuples in which the location domain is non-empty only for HTTP response redirects. The rules for ID passing are as follows:

- If an ID appears in a requested URL, the requested domain learns the ID.
- If an ID appears in the referrer URL, the requested domain and location domain (if it exists) learn the ID.
- If an ID appears in the location URL, the requested domain learns the ID.

We cannot assume that the referrer learns a synced ID appearing in the requested URL or location URL string [38]. In particular, third-party JavaScript executing a sync on a first-party site will cause the first-party to show up as the referrer, even though it may not even be aware of the ID sync. Although we can determine the directionality of ID syncs in the cases of redirects, the fraction of flows in which we could determine both the sender and receiver was small. Hence, when examining cookie synchronization, we focused on which domains knew a given ID, rather than attempting to reconstruct the paths of ID flows.

5.2 Basic results

Before examining the privacy threats that can stem from cookie synchronization, we first provide an overview of cookie syncing activities that occur when browsing under different privacy settings. We ran multiple crawls of the top 3,000 Alexa domains on Amazon EC2¹¹ instances using three different Firefox privacy settings: allowing all cookies (i.e. no privacy-protective measures), allowing all cookies but enabling Do Not Track, and blocking third-party cookies. With all cookies allowed, the impact of Do Not Track on the aggregate statistics we measure was negligible. In particular, enabling Do Not Track only reduced the number of domains involved in synchronization by 2.9% and the number of IDs being synced by 2.6%. This finding is consistent with studies such as Balebako et al. [9] — they find that, due to lack of industry enforcement, Do Not Track provides little practical protection against trackers. We therefore omit further measurement and analysis of the effect of Do Not Track in this section.

Table 4 shows high-level statistics for illustrative crawls under the two third-party cookie settings. We say that an ID is involved in synchronization if it is known by at least two domains. Cookies and domains are involved in synchronization if they contain or know such an ID, respectively. The statistics displayed aggregate both third-party and first-party data, as many domains (e.g. `doubleclick.com`, `facebook.com`) exist in both the Alexa Top 3000 and as third-parties on other sites.

¹¹<http://aws.amazon.com/ec2/>

| Statistic | Third party cookie policy | |
|----------------------------|---------------------------|------------|
| | Allow | Block |
| # IDs | 1308 | 938 |
| # ID cookies | 1482 | 953 |
| # IDs in sync | 435 | 347 |
| # ID cookies in sync | 596 | 353 |
| # (First*) Parties in sync | (407) 730 | (321) 450 |
| # IDs known per party | 1/2.0/1/33 | 1/1.8/1/36 |
| # Parties knowing an ID | 2/3.4/2/43 | 2/2.3/2/22 |

Table 4: Comparison of high-level cookie syncing statistics when allowing and disallowing third-party cookies (top 3,000 Alexa domains). The format of the bottom two rows is minimum/mean/median/maximum. *Here we define a first-party as a site which was visited in the first-party context at any point in the crawl.

Appendix B shows a summary of the top 10 parties involved in cookie synchronization under both cookie policies. Observe that although some parties are involved in less syncing under the stricter cookie policy, many of the top parties receive the same number of IDs. Overall, disabling third-party cookies reduces the number of synced IDs and parties involved in syncing by nearly a factor of two. While this reduction appears promising from a privacy standpoint, in the next section we will see that even with this much sparser amount of data, database merges could enable domains to reconstruct a large portion of a user’s browsing history.

Included in Appendix C is a summary of the top 10 shared IDs under both cookie policies. For a specific example, consider the most shared ID which all third party cookies are allowed, which was originally created by `turn.com`. This ID is created and placed in a cookie on the first page visit that includes Turn as a third-party. On the next page visit, Turn makes GET requests to 25 unique hostnames with a referrer of the form `http://cdn.turn.com/server/ddc.htm?uid=<unique_id>...` that contains its ID. These 25 parties gain knowledge of Turn’s ID, as well as their own tracking cookies, in the process. Similar sharing occurs as the user continues to browse, eventually leading to 43 total domains. With third-party cookies disabled, the top shared IDs come from a disjoint set of parties, largely composed of syncs which share a first party cookie with several third-party sites.

5.3 Back-end database synchronization

We now turn to quantifying how much trackers can learn about users’ browsing histories by merging databases on the back-end based on synced IDs. Cookie syncing allows trackers to associate a given user both with their own pseudonymous ID and with IDs received through syncs, facilitating later back-end merges. We cannot observe these merges directly, so we do not know if such merges occur with any frequency. That said, there is a natural incentive in the tracking ecosystem to aggregate data in order to learn a much larger fraction of a user’s history.

First, assuming no collaboration among third-party trackers, only a handful of trackers are in position to track a sizeable fraction of an individual’s browsing history. As per Olejnik et al [38], if a visited first party appears as the referrer in a request to another domain, we assume the second domain knows about this visit. For a crawl of 3,000 sites when allowing all cookies, only two of the 730 trackers could

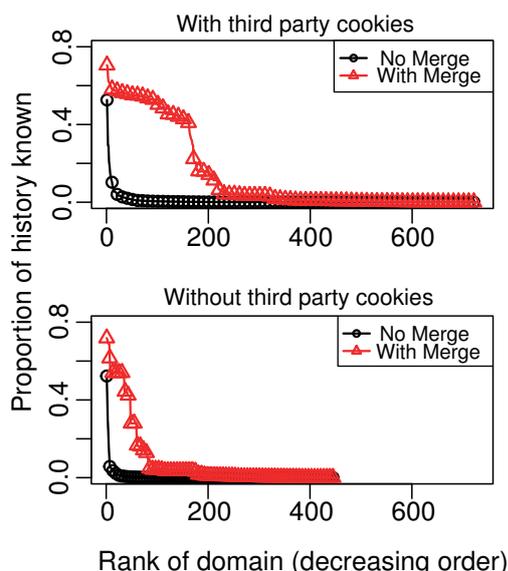


Figure 5: Proportions of user history known when allowing and blocking third party cookies under the two different merging schemes. Note that since the x-axis is sorted by the proportion of a user’s history that a domain can recover, the domains may appear in different orders for the different models.

recover more than 40% of a user’s history and only 11 could recover more than 10%. When disabling third-party cookies, the corresponding numbers are two and six, respectively. These results are consistent with earlier findings in Roesner et al [41].

We consider the following model of back-end database merges: a domain can merge its records with a single other domain that mutually knows some ID. We assume that when two domains merge their records for a particular user, they will share their full records. Our model assumes some collaboration within the tracking ecosystem — among domains already known to share IDs — but is much weaker than assuming full cooperation.

Figure 5 shows the proportion of a user’s 3,000-site browsing history a domain can recover, in decreasing sorted order, if a user enables all cookies. The figure when blocking third-party cookies (also Figure 5) takes an identical shape but is steeper because it only includes roughly 60% as many parties.

Observe that after introducing the ability for a site to merge records directly with one other tracker, the known proportion of a user’s 3,000-site history dramatically increased for a large number of sites. When third-party cookies are allowed, 101 domains can reconstruct over 50% of a user’s history and 161 could recover over 40%. Even when these cookies are blocked, 44 domains could recover over 40% of a user’s history.

Not much is known about how prevalent back-end database merges are. In terms of incentives, a pair of trackers may enter into a mutually beneficial arrangement to increase their respective coverage of users’ browsing histories, or a large tracker may act as a data broker and sell user histories for a fee.

5.4 Respawning and syncing

At a given point in time, cookie synchronization provides a mechanism for trackers to link a user’s history together. Represented as a graph, sites in an individual’s history can be represented as nodes with edges between sites if a user tagged with some pseudonymous ID visited both sites. When a user clears his cookies and restarts browsing, the third parties will place and sync a new set of IDs and eventually reconstruct a new history graph.

Since these history graphs correspond to browsing periods with completely different tracking IDs, they will be disjoint — in other words, trackers can not associate the individual’s history before and after clearing cookies. However, if one of the trackers respawns a particular cookie, parts of the two history graphs can be connected by an edge, thereby linking an individual’s history over time. This inference becomes stronger if this respawned ID is synced to a party present on a large number of the sites that a user visits.

To test this possibility, we ran two 3,000 site crawls on two EC2 instances, A and B. We cleared the cookies, Flash storage, cache, and local storage from machine B and loaded the Flash files from A to seed respawning from Flash. Finally, we ran another 3,000 site crawl on site B.

We discovered a total of 26 domains that respawed IDs between the two crawls on machine B either through Flash or through other means¹². Three of these IDs were later observed in sync flows. After conducting manual analysis, we were unable to determine the exact mechanism through which 18 of these IDs were respawed since we cleared all the storage vectors previously discussed, nor did we detect JavaScript-based browser fingerprinting. We conjecture that these IDs were respawed through some form of passive, server-side fingerprinting¹³.

One of these IDs provides a useful case study. After respawning this ID, its owner, `merchenta.com`, passed it to `adnxs.com` through an HTTP redirect sync call. Now, `merchenta.com` by itself is not in a position to observe a large fraction of a user’s history — it only appears on a single first party domain (`casino.com`). In fact, the largest observed percentage of a user’s history observable by a cookie-respawning domain acting alone was 1.4%. However, by passing its ID to `adnxs.com`, `merchenta.com` enabled a much larger proportion of a user’s history to be linked across state clears.

In particular, we observed `adnxs.com` on approximately 11% of first party sites across the two crawls. Thus `adnxs.com` now has the ability to merge its records for a particular user before and after an attempt to clear cookies, although of course we have no insight into whether or not they actually do so. This scenario enables at least 11% of a user’s history to be tracked over time.

Our measurements in this section illustrate the potential for cookie respawning and syncing event on a single site by a

¹²The exact method here is not important, as we are concerned with the fact that an ID which has been respawed is later involved in sync.

¹³Note that a document from one of these respawning domains, `merchenta.com` mentions tracking by fingerprinting: “Merchenta’s unique fingerprint tracking enables consumers to be engaged playfully, over an extended period of time, long after solely cookie-based tracking loses its effectiveness”, <http://www.merchenta.com/wp-content/files/Merchenta%20Case%20Study%20-%20Virgin.pdf>.

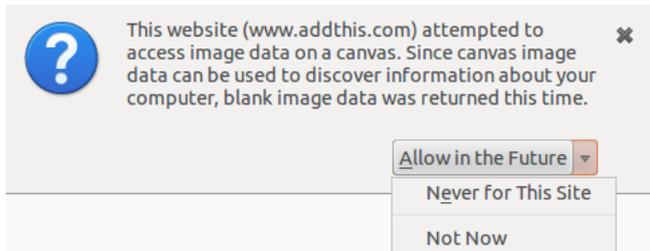


Figure 6: The Tor Browser’s notification dialog for canvas read attempts. The empty image is returned to thwart canvas fingerprinting.

small tracker to enable a large proportion of a user’s history to be tracked by more prolific third parties.

6. DISCUSSION

After presenting an evaluation of advanced tracking techniques, we now discuss the potential defenses against these methods and the implications of our study for privacy-conscious users.

6.1 Mitigation

A blunt way to defend against tracking is to simply block third-party content. This is the approach taken by tools such as Adblock Plus¹⁴ and Ghostery.¹⁵ The user may also disable evercookie storage vectors such as Flash cookies [3], but to the best of our knowledge, tracking vectors such as localStorage, IndexedDB and canvas cannot be disabled, often due to the fact that doing so would break core functionality.

Canvas fingerprinting: The initial canvas fingerprinting study discusses possible countermeasures such as adding noise to the pixel data or trying to produce same pixel results for every system. Finding some barriers to all these options, the paper concludes that asking user permission for each canvas read attempt may be the only effective solution. Indeed, this is precisely the technique adopted in the Tor Browser, the only software that we found to successfully protect against canvas fingerprinting. Specifically, the Tor Browser returns an empty image from all the canvas functions that can be used to read image data [13]. The user is then shown a dialog where she may permit trusted sites to access the canvas. We confirmed the validity of this approach when visiting a site we built which performs browser fingerprinting.

As for more traditional fingerprinting techniques, the Tor browser again appears to be the only effective tool. With the exception of a recent Mozilla effort to limit plugin enumeration [2], browser manufacturers have not attempted to build in defenses against fingerprinting. We note that they are in a position to facilitate such defenses by providing APIs or settings or tools that can be used to develop countermeasures.

Finally, academic studies on mitigating browser fingerprinting are promising but still far from providing practically implementable and comprehensive countermeasures that address all the attack possibilities [10, 36].

¹⁴<https://adblockplus.org>

¹⁵<http://www.ghostery.com>

Evercookies: The straightforward way to defend against evercookies is to clear all possible storage locations. The long list of items removed by the Tor Browser when a user switches to a new identity provides a hint of what can be stored in unexpected corners of the browser: “searchbox and findbox text, HTTP auth, SSL state, OCSP state, site-specific content preferences (including HSTS state), content and image cache, offline cache, Cookies, DOM storage, DOM local storage, the safe browsing key, and the Google wifi geolocation token. . .”[40].

The user interfaces provided by popular browsers for managing browsing information are often fragmented, incomplete, or esoteric. For instance, Firefox’s Clear Recent History interface does not clear localStorage if the user doesn’t select “Everything” as the time range of removal¹⁶ and there is no unified interface for checking what is stored in localStorage and IndexedDB. Similarly, Offline Website Data (AppCache and Cache) can only be checked by visiting a separate `about:cache` page.

Even if the user manages to clear all storage vectors, the fact that Flash storage is not isolated¹⁷ between browsers which use the Adobe Flash plugin (e.g. Firefox, Chromium, and Internet Explorer) still creates an opportunity for respawning. Consider the common scenario of a multi-user environment where Alice uses browser A and Bob uses browser B, without any OS-level separation of user accounts. Assume that Alice is privacy-conscious and clears browser state frequently, but Bob does not. Consider an ID on Browser A is shared between Browser A’s Flash Cookies and HTTP Cookies. When Bob browses, X may be respawning as an HTTP cookie in browser B. In Section 4.2, we showed that this behavior occurs in the wild. Now when Alice completely clears the state of Browser A, the ID X will be removed from common flash storage and Browser A’s HTTP storage. Crucially, however, when Bob browses again, it could be respawning from B’s HTTP storage to common flash storage and later when Alice browses again, back to A’s HTTP storage. We showed in Section 4.3 that HTTP-to-Flash respawning occurs in the wild as well. Thus the only way to defend against this attack in a multi-browser environment is to clear state on all browsers simultaneously. As a proof-of-concept, we manually tested the first-party domains on which we observe HTTP-to-Flash respawning (Appendix Table 8) and we found this exact scenario occurs on both `casino.com` and `xlovecam.com`.

Cookie syncing: We’re not aware of any tools that specifically block cookie syncing. The bluntest approach, of course, is to simply block third-party cookie placement and HTTP traffic. EFF’s newly released tool Privacy Badger¹⁸ uses heuristics to block third-party cookies with the goal of preventing third-party tracking, erring on the side of false positives (i.e., blocking too many cookies). The Tor Browser Bundle (TBB) prevents cross-site cookie tracking by disabling all third-party cookies, and not storing any persistent data such as cookies, cache or localStorage. A more targeted solution would be to block third-party traffic containing strings that are cookie values, but this approach will

¹⁶Bug 527667 https://bugzilla.mozilla.org/show_bug.cgi?id=527667

¹⁷Confirmed through manual analysis

¹⁸<https://www.eff.org/privacybadger>

likely suffer from false negatives. However, even a perfect blocking tool is flawed if it is not used immediately from a completely fresh browsing state. For instance, if a user browses for a short amount of time before installing such a tool, trackers may have already placed and synced cookies — enabling them to merge data in the back-end. If these IDs are maintained through a hard-to-block technique such as canvas fingerprinting, the trackers can still follow a user as he browses and link their records through these previously-established syncing relationships even if all future syncs are blocked.

6.2 The effect of opt-out

In order to study the effect of ad-industry opt-out tools on the tracking mechanisms we study, we opted-out from all the listed companies on the Network Advertising Initiative (NAI)¹⁹ and European Interactive Digital Advertising Alliance (EDAA)²⁰ opt-out pages.

Canvas fingerprinting: For each canvas fingerprinting script we visited two sites that included this script. We did not observe any website that stopped collecting canvas fingerprint due to opt-out.²¹ This was despite the fact that AddThis was listed on the NAI opt-out page and Ligatus (second most popular canvas fingerprinter) was listed on EDAA’s page.

We also tried opting-out by on AddThis’ own Data Collection Opt-Out website²², which again, did not stop AddThis’s script collecting the canvas fingerprint.

Respawning: We did not observe any change in cookie respawning from HTTP to Flash cookies. This is expected as the parties involved are not participants in the advertising opt-out initiatives.

Cookie syncing: The use of opt-out cookies reduces the number of IDs involved in cookie synchronization by 30%. However, we see only a 5% reduction in the number of parties involved in synchronization. This reduction is comparatively smaller than the reduction seen when the browser is set to block third-party cookies. The composition of the top parties involved in synchronization is nearly the same as in the first-party cookie only case seen in Appendix B. In Section 5.3 we show how, even under the larger reduction in sync activity afforded by blocking all third-party cookies, it is possible to recover a large portion of a user’s browsing history using just a small number of the parties involved.

Note that most companies offering or honoring the opt-outs we evaluated do not promise to stop tracking when a user opts out, but only behavioral advertising. While we observed tiny or nonexistent reductions in various forms of tracking due to opt-out, we make no claims about how opt-outs affect behavioral advertising.

¹⁹<http://www.networkadvertising.org/choices/>

²⁰<http://www.youronlinechoices.com/uk/your-ad-choices>

²¹We observed that two of the 20 fingerprinting scripts (revtrax.com and vcmedia.vn) were missing on the sites we found them before, though we checked to ensure that this was not related to opt-out.

²²<http://www.addthis.com/privacy/opt-out>

6.3 Implications

Let us consider the level of user effort and sophistication required for effective mitigation. First, users must be very careful in their use of existing tools, such as clearing state on all browsers at once or installing blocking tools before cookie syncing has occurred. Second, users must accept usability drawbacks such as the prompt for Canvas API access. Third, there are also trade-offs in functionality and content availability. Finally, the rapid pace at which new tracking techniques are developed and deployed implies that users must constantly install and update new defensive tools. It is doubtful that even privacy-conscious and technologically-savvy users can adopt and maintain the necessary privacy tools without ever experiencing a single misstep.

Evercookies were at the center of fierce debates when Soltani et al. reported their findings [43] a few years ago. Although this resulted in a lawsuit and a \$500,000 settlement [14], we find an increasing number of websites using these tracking technologies as well as significant advances in the technologies themselves.

The World Wide Web Consortium (W3C) standards documents that describe three new storage APIs (localStorage, IndexedDB and WebStorage APIs) have the same boilerplate warning about the tracking potentials of these mechanisms²³ and mention the necessity of an interface to communicate the evercookie risk. Perhaps a fruitful future direction for standards bodies is to consider privacy issues at the design stage, acknowledging that without such a proactive effort, tracking techniques are likely to have the upper hand over defenses. W3C’s draft specification “Fingerprinting Guidance for Web Specification Authors” is a notable effort in this direction, for providing a guideline to Web specification authors about privacy risks of browser fingerprinting [15].

6.4 A Path Forward

Blocking tools are currently the primary solution to third-party tracking for the informed user. We believe that these tools can be greatly improved by a back-end consisting of regular web-scale crawls. Crawlers can incorporate sophisticated rules to detect unwanted tracking, as we have shown, whereas it would be difficult to deploy these directly into browser tools. Accordingly, we plan to further scale our crawling infrastructure, while continuing to release results in a machine-readable format.

Crawler-supported blocking tools could also benefit from machine learning and crowd-sourcing (instead of rules hand-coded by experts) for minimizing false positives and negatives. For example, we have produced an initial classification of canvas fingerprinting scripts on 100,000 sites, but there are surely many more such scripts in the web’s long tail, which suggests that a semi-supervised learning approach could be effective. The resulting classifier would label scripts that access the canvas API as canvas fingerprinters or non-canvas-fingerprinters. Turning to crowdsourcing, a browser tool could default to blocking all canvas write/read attempts, but slowly incorporate user feedback about broken functionality to train a model for identifying true fin-

²³<http://www.w3.org/TR/webstorage/#user-tracking>,
<http://www.w3.org/TR/IndexedDB/#user-tracking>,
<http://www.w3.org/TR/webdatabase/#user-tracking>

gerprinting attempts. Of course, these two approaches can be combined.

Finally, publishers have little insight into the types of tracking occurring on their own sites. The tools that we and others have built can be re-purposed to provide transparency not just to end-users but also allow publishers an in-depth look into how trackers collect data from their sites, where the data flows, and how it is used. This will allow them to discriminate between advertising or analytics providers on the basis of privacy practices.²⁴ If combined with public pressure to hold *first parties* accountable for online tracking and not just third parties, it can move online tracking in a more transparent and privacy-friendly direction.

7. CONCLUSION

We present a large-scale study of tracking mechanisms that misuse browser features to circumvent users' tracking preferences. We employed innovative measurement methods to reveal their prevalence and sophistication in the wild. Current options for users to mitigate these threats are limited, in part due to the difficulty of distinguishing unwanted tracking from benign behavior. In the long run, a viable approach to online privacy must go beyond add-ons and browser extensions. These technical efforts can be buttressed by regulatory oversight. In addition, privacy-friendly browser vendors who have hitherto attempted to take a neutral stance should consider integrating defenses more deeply into the browser.

8. ACKNOWLEDGEMENTS

The authors would like to thank Joseph Bonneau, Edward Felten, Georg Koppen, Lukasz Olejnik, Mike Perry, Vitaly Shmatikov, Roland Illig, and Matthew Wright for valuable feedback, Dillon Reisman and Pete Zimmerman for helping develop some of the infrastructure used, Oscar Reparaz for chroot tips and Junjun Chen for earlier work on cookie syncing that helped our understanding of the practice. For KU Leuven, this work was partially funded by the projects IWT SBO SPION, FWO G.0360.11N, FWO G.0686.11N, and the KU Leuven BOF OT project ZKC6370 OT/13/070.

9. REFERENCES

- [1] Privacychoice - get a free privacy scan of your site. <http://privacychoice.org/assessment>.
- [2] Bug 757726 - disallow enumeration of navigator.plugins. https://bugzilla.mozilla.org/show_bug.cgi?id=757726, May 2012.
- [3] Manage, disable Local Shared Objects | Flash Player. <http://helpx.adobe.com/flash-player/kb/disable-local-shared-objects-flash.html>, 2014.
- [4] Doubleclick ad exchange real-time bidding protocol: Cookie matching. <https://developers.google.com/ad-exchange/rtb/cookie-guide>, February 2014.
- [5] Selenium - Web Browser Automation. <http://docs.seleniumhq.org/>, 2014.
- [6] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel. FPDetective: Dusting the Web for fingerprinters. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1129–1140. ACM, 2013.
- [7] M. Ayenson, D. J. Wambach, A. Soltani, N. Good, and C. J. Hoofnagle. Flash cookies and privacy II: Now with HTML5 and ETag respawning. *World Wide Web Internet and Web Information Systems*, 2011.
- [8] M. Backes, A. Kate, M. Maffei, and K. Pecina. Obliviad: Provably secure and practical online behavioral advertising. In *IEEE Security and Privacy (S&P)*, pages 257–271. IEEE, 2012.
- [9] R. Balebako, P. Leon, R. Shay, B. Ur, Y. Wang, and L. Cranor. Measuring the effectiveness of privacy tools for limiting behavioral advertising. In *Web 2.0 Workshop on Security and Privacy (W2SP)*. IEEE, 2012.
- [10] F. Besson, N. Bielova, T. Jensen, et al. Enforcing Browser Anonymity with Quantitative Information Flow. 2014.
- [11] M. Bilenko, M. Richardson, and J. Y. Tsai. Targeted, not tracked: Client-side solutions for privacy-friendly behavioral advertising. In *Privacy Enhancing Technologies (PETS)*. Springer, 2011.
- [12] P. E. Black. Ratcliff/Obershelp pattern recognition. <http://xlinux.nist.gov/dads/HTML/ratcliffObershelp.html>, December 2004.
- [13] K. Brade. gitweb.torproject.org - torbrowser.git/blob - src/current-patches/firefox/0019-add-canvas-image-extraction-prompt.patch. <https://gitweb.torproject.org/torbrowser.git/blob/HEAD:/src/current-patches/firefox/0019-Add-canvas-image-extraction-prompt.patch>, November 2012.
- [14] W. Davis. KISSmetrics Finalizes Supercookies Settlement. <http://www.mediapost.com/publications/article/191409/kissmetrics-finalizes-supercookies-settlement.html>, 2013. [Online; accessed 12-May-2014].
- [15] N. Doty. Fingerprinting Guidance for Web Specification Authors. <http://w3c.github.io/fingerprinting-guidance/>, 2014.
- [16] P. Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies (PETS)*, pages 1–18. Springer, 2010.
- [17] C. Eubank, M. Melara, D. Perez-Botero, and A. Narayanan. Shining the floodlights on mobile web tracking - a privacy survey. In *"Web 2.0 Security and Privacy"*, May 2013.
- [18] E. W. Felten. If You're Going to Track Me, Please Use Cookies. <https://freedom-to-tinker.com/blog/felten/if-youre-going-track-me-please-use-cookies/>, 2009.
- [19] M. Fredrikson and B. Livshits. Repriv: Re-imagining content personalization and in-browser privacy. In *IEEE Security and Privacy (S&P)*, pages 131–146. IEEE, 2011.
- [20] S. Guha, B. Cheng, and P. Francis. Privad: practical privacy in online advertising. In *USENIX Conference*

²⁴In fact, there is a fledgling commercial market for such tools [1], but they are not very sophisticated.

- on *Networked Systems Design and Implementation*, pages 169–182. USENIX Association, 2011.
- [21] S. Kamkar. Evercookie - virtually irrevocable persistent cookies. <http://samy.pl/evercookie/>, Sep 2010.
- [22] M. Kerrisk. strace(1) - linux manual page. <http://man7.org/linux/man-pages/man1/strace.1.html>, May 2014.
- [23] T. Kohno, A. Broido, and K. C. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.
- [24] R. Kotcher, Y. Pei, P. Jumde, and C. Jackson. Cross-origin pixel stealing: timing attacks using CSS filters. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1055–1062. ACM, 2013.
- [25] B. Krishnamurthy and C. Wills. Privacy diffusion on the Web: a longitudinal perspective. In *International Conference on World Wide Web*, pages 541–550. ACM, 2009.
- [26] B. Krishnamurthy and C. E. Wills. On the leakage of personally identifiable information via online social networks. In *ACM Workshop on Online Social Networks*, pages 7–12. ACM, 2009.
- [27] B. Liu, A. Sheth, U. Weinsberg, J. Chandrashekar, and R. Govindan. AdReveal: Improving transparency into online targeted advertising. In *ACM Workshop on Hot Topics in Networks*, page 12. ACM, 2013.
- [28] J. Mayer. Tracking the trackers: Self-help tools. <https://cyberlaw.stanford.edu/blog/2011/09/tracking-trackers-self-help-tools>, September 2011.
- [29] J. R. Mayer and J. C. Mitchell. Third-party web tracking: Policy and technology. In *IEEE Symposium on Security and Privacy (S&P)*, pages 413–427. IEEE, 2012.
- [30] A. M. McDonald and L. F. Cranor. Survey of the Use of Adobe Flash Local Shared Objects to Respawn HTTP Cookies, A. *ISJLP*, 7:639, 2011.
- [31] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham. Fingerprinting information in JavaScript implementations. In *Web 2.0 Workshop on Security and Privacy (W2SP)*, volume 2. IEEE, 2011.
- [32] K. Mowery and H. Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In *Web 2.0 Workshop on Security and Privacy (W2SP)*. IEEE, 2012.
- [33] M. Mulazzani, P. Reschl, M. Huber, M. Leithner, S. Schrittwieser, E. Weippl, and F. C. Wien. Fast and reliable browser identification with JavaScript engine fingerprinting. In *Web 2.0 Workshop on Security and Privacy (W2SP)*, volume 1. IEEE, 2013.
- [34] A. Narayanan, J. Mayer, and S. Iyengar. Tracking Not Required: Behavioral Targeting. <http://33bits.org/2012/06/11/tracking-not-required-behavioral-targeting/>, 2012.
- [35] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. You are what you include: Large-scale evaluation of remote javascript inclusions. In *ACM Conference on Computer and Communications Security (CCS)*, pages 736–747. ACM, 2012.
- [36] N. Nikiforakis, W. Joosen, and B. Livshits. PriVaricator: Deceiving Fingerprinters with Little White Lies. Available at <http://research.microsoft.com/en-us/um/people/livshits/papers%5Ctr%5Cprivaricator.pdf>.
- [37] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *IEEE Symposium on Security and Privacy*, pages 541–555. IEEE, 2013.
- [38] L. Olejnik, T. Minh-Dung, and C. Castelluccia. Selling Off Privacy at Auction. In *Annual Network and Distributed System Security Symposium (NDSS)*. IEEE, 2014.
- [39] C. R. Orr, A. Chauhan, M. Gupta, C. J. Frisz, and C. W. Dunn. An approach for identifying JavaScript-loaded advertisements through static program analysis. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 1–12. ACM, 2012.
- [40] M. Perry, E. Clark, and S. Murdoch. The design and implementation of the Tor browser [draft]. <https://www.torproject.org/projects/torbrowser/design>, 2013.
- [41] F. Roesner, T. Kohno, and D. Wetherall. Detecting and Defending Against Third-Party Tracking on the Web. In *Symposium on Networking Systems Design and Implementation*. USENIX Association, 2012.
- [42] N. Singer. Do Not Track? Advertisers Say ‘Don’t Tread on Us’. <http://www.nytimes.com/2012/10/14/technology/do-not-track-movement-is-drawing-advertisers-fire.html>, 2012.
- [43] A. Soltani, S. Canty, Q. Mayo, L. Thomas, and C. J. Hoofnagle. Flash Cookies and Privacy. In *AAAI Spring Symposium: Intelligent Information Privacy Management*. AAAI, 2010.
- [44] O. Sorensen. Zombie-cookies: Case studies and mitigation. In *Internet Technology and Secured Transactions (ICITST)*, pages 321–326. IEEE, 2013.
- [45] P. Stone. Pixel perfect timing attacks with HTML5. *Context Information Security (White Paper)*, 2013.
- [46] A. Taly, J. C. Mitchell, M. S. Miller, J. Nagra, et al. Automated analysis of security-critical javascript apis. In *IEEE Security and Privacy (S&P)*, pages 363–378. IEEE, 2011.
- [47] J. Temple. Stale Cookies: How companies are tracking you online today. <http://blog.sfgate.com/techchron/2013/10/02/stale-cookies-how-companies-are-tracking-you-online-today/>, 2013.
- [48] M. Tran, X. Dong, Z. Liang, and X. Jiang. Tracking the trackers: Fast and scalable dynamic analysis of web content for privacy violations. In *Applied Cryptography and Network Security*, pages 418–435. Springer, 2012.
- [49] M.-D. Tran, G. Acs, and C. Castelluccia. Retargeting without tracking. *arXiv preprint arXiv:1404.4533*, 2014.
- [50] T. Unger, M. Mulazzani, D. Fruhwirt, M. Huber, S. Schrittwieser, and E. Weippl. SHPF: Enhancing HTTP(S) Session Security with Browser Fingerprinting. In *Availability, Reliability and Security (ARES)*, pages 255–261. IEEE, 2013.

[51] V. Vasilyev. Valve/fingerprintjs.
<https://github.com/Valve/fingerprintjs>, 2012.

APPENDIX

A. FLASH COOKIES WITH THE MOST RESPAWNS

| Flash domain | # respawned cookies | |
|-----------------------------|---------------------|--------|
| | Pass 1 | Pass 2 |
| bbcdn-bbnaut.ibillboard.com | 63 | 69 |
| irs01.net | 21 | 18 |
| embed.wistia.com | 14 | 13 |
| source.mmi.bemobile.ua | 13 | 14 |
| kiks.yandex.ru | 11 | 11 |
| static.baifendian.com | 10 | 10 |
| tv.sohu.com | 7 | 7 |
| ar.hao123.com | 3 | 2 |
| embed-ssl.wistia.com | 3 | 3 |
| img5.uloz.to | 3 | 3 |

Table 5: The Flash cookies that respawn most cookies on Alexa top 10,000 sites. The rightmost two columns represent the number of cookies respawned in two crawls made with the same set of Flash cookies ($Crawl_{2,3}$).

B. TOP PARTIES INVOLVED IN COOKIE SYNC

| All Cookies Allowed | | No 3P Cookies | |
|---------------------|-------|-----------------|-------|
| Domain | # IDs | Domain | # IDs |
| gemius.pl | 33 | gemius.pl | 36 |
| doubleclick.net | 32 | 2o7.net | 27 |
| 2o7.net | 27 | omtrdc.net | 27 |
| rubiconproject.com | 25 | cbsi.com | 26 |
| omtrdc.net | 24 | parsely.com | 16 |
| cbsi.com | 24 | marinsm.com | 14 |
| adnxs.com | 22 | gravity.com | 14 |
| openx.net | 19 | cxense.com | 13 |
| cloudfront.net | 18 | cloudfront.net | 10 |
| rlcdn.com | 17 | doubleclick.net | 10 |

Table 6: Number of IDs known by the Top 10 parties involved in cookie sync under both the policy of allowing all cookies and blocking third-party cookies.

C. TOP IDS INVOLVED IN COOKIE SYNC

| All Cookies Allowed | | No 3P Cookies | |
|---------------------|------|----------------------|------|
| ID Creator | # D. | ID Creator | # D. |
| turn.com | 43 | sociomantic.com | 22 |
| adsvr.org | 30 | mybuys.com | 11 |
| mookie1.com | 29 | mybuys.com | 11 |
| Unknown* | 24 | mercadolibre.com | 9 |
| media6degrees.com | 23 | shinobi.jp | 7 |
| parsely.com | 22 | newsanalytics.com.au | 6 |
| Unknown* | 19 | microsoft.com | 6 |
| titaltv.com | 18 | mercadolibre.cl | 5 |
| crwdcntrl.net | 18 | mercadolibre.com.ar | 5 |
| uservice.com | 15 | rackspace.com | 5 |

Table 7: Number of domains which have knowledge of unique IDs created by each listed domain. ID creator determined manually by first placement of cookie (* the relationship was unclear from HTTP/cookie logs).

D. LIST OF HTTP RESPAWNING SCRIPTS

| First-Party Domains | Source of Respawn | Script Source |
|--|--|---|
| accountonline.com (citi.com), fling.com*, flirt4free.com, zoosk.com | Third-party: Iovation Fraud Detection | https://mpsnare.iesnare.com/snare.js https://mpsnare.iesnare.com/stmgwb2.swf |
| seoprofiler.com, seobook.com, bi- grock.in, imperiaonline.org, me- diatemple.net, resellerclub.com | First-party: Post Affiliate Pro Software | http://seobook.com/aff/scripts/trackjs.js |
| twitch.tv, justin.tv | Third-party: Shared CDN | http://www-cdn.jtvnw.net/assets/global- 6e555e3e646ba25fd387852cd97c19e1.js |
| casino.com | First-party: Unknown/In-house | http://www.casino.com/shared/js/mts.tracker.js |
| xlovecam.com | First-party: Unknown/In-house | http://www.xlovecam.com/colormaker.js |

Table 8: Summary of HTTP respawning. “Source of Respawn” describes whether or not the tracking occurs in the first-party or third-party context and lists the entity responsible for writing the script. * Interestingly fling.com has the ID passed from the third-party context and saved in the first-party context

E. LIST OF CANVAS FINGERPRINTING SCRIPTS

| Domain | URL of the Fingerprinting Script |
|----------------------------|---|
| addthis.com | http://ct1.addthis.com/static/r07/core130.js, http://ct1.addthis.com/static/r07/sh157.html# and 16 others |
| ligatus.com | http://i.ligatus.com/script/fingerprint.min.js |
| kitcode.net | http://src.kitcode.net/fp2.js |
| vcmedia.vn | http://admicro1.vcmedia.vn/fingerprint/figp.js |
| amazonaws.com ¹ | https://s3-ap-southeast-1.amazonaws.com/af-bdaz/bquery.js |
| shorte.st | http://static.shorte.st/js/packed/smeadvert-intermediate-ad.js?v1.7.10 |
| ringier.cz | http://stat.ringier.cz/js/fingerprint.min.js |
| cya2.net | http://cya2.net/js/STAT/89946.js?ver=adl&cid=T... |
| revtrax.com | http://images.revtrax.com/RevTrax/js/fp/fp.min.jsp |
| pof.com | http://www.pof.com/ |
| rackcdn.com ² | https://c44ed9b5e0739c3-dcbf3c0901f34702b963a7ca35c5bc1c.ssl.cf2.rackcdn.com/mongoose.fp.js |
| hedyera.com | http://www.hedyera.com/js/dota/dota.js |
| meinkauf.at | http://www.meinkauf.at/assets/application-74bbc9cea66102ea5766faa9209cf3e0.js |
| freevoipdeal.com | http://www.freevoipdeal.com/en/asset/js/39b4e838c58e140741f9752542545e77 |
| voipbuster.com | http://www.voipbuster.com/en/asset/js/8ecf64add423a396f83430f9357a0e55 |
| nonoh.net | http://www.nonoh.net/asset/js/e4cf90bfdfa29f5fd61050d14a11f0a1 |
| 49winners.com | http://49winners.com/js/49w3/fingerprint.js?v=1.1 |
| freecall.com | http://www.freecall.com/asset/js/f4ccb1cb0e4128b6d4b08f9eb2c8deb4 |
| domainsigma.com | http://static.domainsigma.com/static/public/js/common.9b6f343c.js |
| insnw.net ³ | http://dollarshaveclub-002.insnw.net/assets/dsc/dsc.fingerprint-b01440d0b6406b266f8e0bd07c760b07.js |

Table 9: URLs of Canvas Fingerprinting JavaScript. The URL parameters snipped for brevity are denoted by ...

1: s3-ap-southeast-1.amazonaws.com (sends the collected fingerprint to adsfactor.net domain).

2: 44ed9b5e0739c3dcbf3c0901f34702b963a7ca35c5bc1c.ssl.cf2.rackcdn.com (sends the collected fingerprint to api.gonorthleads.com). 3:dollarshaveclub002.insnw.net