Cookie Swap Party: Abusing First-Party Cookies for Web Tracking

Quan Chen
qchen10@ncsu.edu
North Carolina State University
Raleigh, USA

Panagiotis Ilia
pilia@uic.edu
University of Illinois at Chicago
Chicago, USA

Michalis Polychronakis
mikepo@cs.stonybrook.edu
Stony Brook University
Stony Brook, USA

Alexandros Kapravelos
akaprav@ncsu.edu
North Carolina State University
Raleigh, USA

ABSTRACT

As a step towards protecting user privacy, most web browsers perform some form of third-party HTTP cookie blocking or periodic deletion by default, while users typically have the option to select even stricter blocking policies. As a result, web trackers have shifted their efforts to work around these restrictions and retain or even improve the extent of their tracking capability.

In this paper, we shed light into the increasingly used practice of relying on *first-party* cookies that are set by *third-party* JavaScript code to implement user tracking and other potentially unwanted capabilities. Although unlike third-party cookies, first-party cookies are not sent automatically by the browser to third-parties on HTTP requests, this tracking is possible because any included third-party code runs in the context of the parent page, and thus can fully set or read existing first-party cookies—which it can then leak to the same or other third parties. Previous works that survey user privacy on the web in relation to cookies, third-party or otherwise, have not fully explored this mechanism. To address this gap, we propose a dynamic data flow tracking system based on Chromium to track the leakage of first-party cookies to third parties, and used it to conduct a large-scale study of the Alexa top 10K websites. In total, we found that 97.72% of the websites have first-party cookies that are set by third-party JavaScript, and that on 57.66% of these websites there is at least one such cookie that contains a unique user identifier that is diffused to multiple third parties. Our results highlight the privacy-intrusive capabilities of first-party cookies, even when a privacy-savvy user has taken mitigative measures such as blocking third-party cookies, or employing popular crowd-sourced filter lists such as EasyList/EasyPrivacy and the Disconnect list.

ACM Reference Format:
Quan Chen, Panagiotis Ilia, Michalis Polychronakis, and Alexandros Kapravelos. 2021. Cookie Swap Party: Abusing First-Party Cookies for Web Tracking. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3442381.3449837

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449837

1 INTRODUCTION

Most of the JavaScript (JS) [8] code on modern websites is provided by external, third-party sources [18, 26, 31, 38]. Third-party JS libraries execute in the context of the page that includes them and have access to the DOM interface of that page. In many scenarios it is preferable to allow third-party JS code to run in the context of the parent page. For example, in the case of analytics libraries, certain user interaction metrics (e.g., mouse movements and clicks) cannot be obtained if JS code executes in a separate `iframe`.

This cross-domain inclusion of third-party JS code poses security and privacy risks. In the web context, third-party tracking is arguably the most persistent intrusion to user privacy. Multiple previous efforts [21, 25, 29, 32, 34] attempt to understand such tracking. Browser vendors are also beginning to restrict third-party tracking: Firefox by default blocks HTTP cookies set for known trackers [2], and Safari introduced Intelligent Tracking Prevention (ITP) that uses machine learning techniques to identify trackers [9]. However, these efforts largely focused on web tracking that utilizes third-party cookies, and not much attention has been paid on exploring how *first-party* cookies are being used in regard to web tracking. Since third-party scripts that are included in the page do not have to adhere to the restrictions of the *Same Origin Policy* (SOP) [28], they can set first-party cookies on behalf of the third-party tracker. On subsequent visits, the same JS code (or code by other third parties that runs in the page) can read the cookies and transmit them back to the trackers.

One of the first works on web tracking, by Roesner et al. [34], explored the case of *within-site* tracking where a third-party script, such as Google Analytics, sets a first-party cookie and then sends its value back to the third-party's server. However, that work only explored the case where the first-party cookie is sent back to the domain from which the script originated, but not cases where other different third parties are involved in receiving the cookie's value. A recent work by Fouad et al. [23] explores the case where a first-party cookie's value is included in a request to a third party domain, and characterize this behavior as *"first to third party cookie syncing,"* but they do not distinguish the origin of the diffused cookies, i.e., whether they are set by third-party code. Since previous works have not fully explored first-party cookies with respect to their use in web tracking, in this work we aim at bridging this gap by shedding light into how third parties are currently leveraging first-party cookies to coordinate their efforts and share identifiers to track users.

Specifically, we conducted a large-scale analysis of the Alexa top 10K websites aimed at measuring the prevalence of web tracking that

is based on first-party cookies set by third-party scripts, which we hereafter refer to as *external cookies*. To do so, we utilize an analysis system that observes the data flows of JS code at runtime using dynamic taint analysis. This system automatically marks as tainted the cookies that are written to `document.cookie` by code that has a different origin than the current first-party context. When such cookies are accessed by JS code, dynamic taint tracking makes sure to propagate the taint to any values that are derived from the tainted cookies, and as the tainted values reach network sinks, such as `XMLHttpRequest`, an alert is raised and the event is logged.

In total, our analysis shows that 9,772 (97.72%) of the Alexa top 10K websites have first-party cookies that are set by third-party JavaScript code, and that alarmingly, 57.66% of these websites have at least one such cookie that contains a unique user identifier (i.e., tracking ID) that is diffused, whether by the original script that set the cookie or by scripts from a different third-party, to third-party destinations that are *different* than the party from which the cookie originated. This analysis is enabled by two additional core contributions of this paper: (i) we leverage and improve the methods proposed by previous works [21, 22] to automatically detect first-party cookies, set by third-party JS, whose values contain information that uniquely identifies a user (which we refer to as UID-containing cookies), and (ii) we develop heuristics that match the values of UID-containing cookies with parts of the outgoing requests that contain tainted values, in order to identify information flows between the third party that initially set the cookies, and the third parties that receive information derived from those cookies. Our results demonstrate the privacy risk posed by external cookies, even when popular mitigation techniques, such as completely blocking third-party cookies, or crowd-sourced filter lists (e.g., EasyList, EasyPrivacy) are employed, and motivate the need for more effective privacy protection countermeasures.

We summarize our major contributions as follows:

- We bridge an important gap that has been largely neglected in previous research by shedding light into how *first-party* cookies that are set by third-party JavaScript code, which we refer to as *external cookies*, are currently being used in web tracking scenarios.
- We leverage data flow analysis techniques that give us detailed insights into the runtime behaviors of JavaScript code with respect to whether, and if so, how, external cookies are being utilized by third-party JavaScript code.
- We conduct a large-scale study of the Alexa top 10K websites to assess the prevalence of external cookies and explore whether they are being used for tracking purposes.
- We develop techniques to automatically detect tracking ID cookies, and identify information flows between third parties that exchange tracking IDs. We find that external cookies are already being actively used on a large majority of the Alexa top 10K websites to facilitate web tracking.

2 BACKGROUND

2.1 HTTP Cookies

HTTP cookies can be categorized as first-party or third-party, depending on their domain of origin. The cookies set when visiting a website are considered as first-party, while those set by other domains as a result of loading external resources are considered as third-party. Consequently, if the same third-party resource (e.g., a popular JavaScript library) is present on multiple websites, it enables cross-site tracking: any third-party domain that host resources referenced by multiple websites can track users across these sites.

Recognizing the potential for this privacy abuse, all major browsers provide a way to block third-party cookies, with some browsers even blocking them by default [1]. Furthermore, an experimental policy that blocks cookies from known tracker domains has recently been introduced in Firefox [2]. Safari also implements Intelligent Tracking Prevention (ITP) that uses machine learning to classify whether a domain is likely to be a tracker, and block its cookies [9].

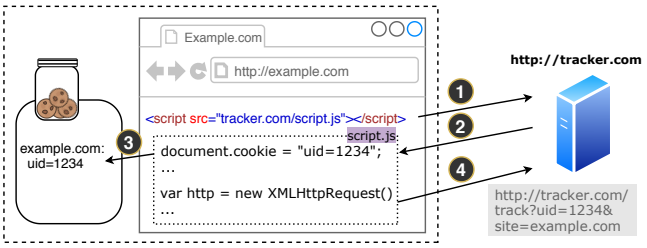

Figure 1: Example of *within-site* tracking [34], as used by web analytics services. Third-party code stores UID by setting an *external* cookie (3), which is later sent back to the same third-party (4).

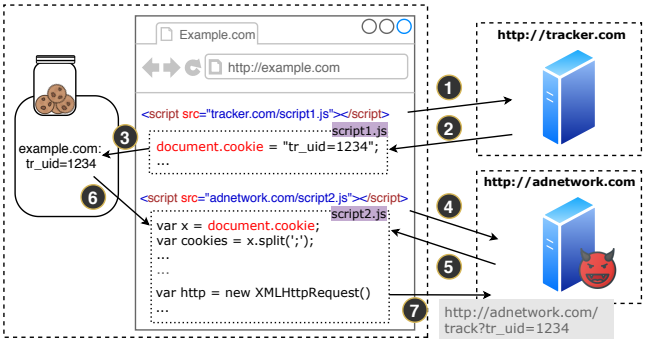

Figure 2: Example of UID sharing between two third parties. One third party reads the external cookie (6) that was set by another third party (3), and "steal" it (7).

2.2 Accessing Cookies from JavaScript

Besides including cookies in the HTTP `Set-Cookie` header field, another way to set cookies in the user's browser is to do so programmatically via JavaScript, through the `document.cookie` property provided by the Document Object Model (DOM). We note that `document.cookie` is the only interface that JavaScript has to the cookies of a website, except those offered by the browser's extension APIs (e.g., Chrome's `chrome.cookies` [3]) that are *only* accessible to browser extensions, and which we do not consider in this paper. Our focus here is how third parties that already have their JS code embedded in first-party websites can abuse this access to track users via cookies set by the embedded code.

2.3 Evading Third-Party Cookie Policies

Instead of using traditional third-party cookies, online trackers can simply have their JavaScript code execute in the first-party context and use `document.cookie` to set first-party *external* cookies. In contrast to third-party cookies that can be easily blocked or deleted without affecting the functionality and usability of the website, this is not so straightforward in the case of first-party cookies.

In Figure 1 we present the common case of third-party services that have code running in the first-party website and utilizing external cookies. In this case, the third-party code sets an external cookie ❸ (note that the cookie is set for the first-party domain) and sends its value back to the third party from which the code originated ❹. This tracking method, which only allows tracking a user returning to a previously visited website, is typically employed by web analytics services, such as Google Analytics.

Although this approach may affect user privacy, as it can track the visits of a user to the first-party website, and possibly artifacts of the user's behavior while navigating the page, it can be considered as benign, or at least acceptable. However, this is not the case when the value of an external cookie is derived based on fingerprinting, when it encodes information that can be used to link the user to a specific user profile (e.g., the user's IP address), and when the same values are being set as both a first-party and a third-party cookie.

The most important and interesting case is when the external cookie is leaked to a third party that is *different* from the one that previously set that cookie. This could be either the result of a co-operation between the two third-party services, or the result of a shady third party that uses its own code to read the cookies that were set by other third parties. As depicted in Figure 2, the code from `http://tracker.com` sets an external cookie (`tr_uid`) ❸, and then different code that is fetched from another third-party domain (`script2.js` from `http://adnetwork.com`) ❺ reads all the first-party cookies of the website ❻, extracts the value of `tr_uid`, and sends it to `http://adnetwork.com` ❼, essentially "stealing" the external cookie. To make matters worse, it is quite possible to have a large number of third parties being involved in reading those external cookies and exchanging identifiers between them.

3 METHODOLOGY

3.1 Challenges in Measuring External Cookies

There are many technical challenges involved in accurately measuring and understanding the usage of external cookies. First, since external cookies are *not* set upon receiving an HTTP response with a `Set-Cookie` header, but rather by third-party JS code running in the first-party context, it is not possible to distinguish external cookies from other first-party cookies simply by observing the network traffic. Thus, the measurement system needs to keep track of the origins of each JS unit executing on the page, and be able to attribute the setting of first-party cookies to the initiating script. That is, it needs to hook the JS interface for updating the cookie storage of a website, identify the initiator script domain responsible for setting cookies through this interface and, if the initiator's domain is different from the first-party domain, record the cookies being set.

Second, unlike traditional third-party cookies, external cookies are also *not* sent back to third parties encoded verbatim in the HTTP request header, but are instead read back and sent by JS code, which can potentially apply arbitrary transformations (e.g., encryption or hashing) before sending them. Therefore, the JS cookie storage interface also needs to be hooked for read accesses. Considering that the read-back values from this interface might contain both external cookies and non-external ones (i.e., first-party cookies *not* set by third-party JS code), the measurement system needs a way to attach this information to the read-back values and track any values derived from external cookies. Furthermore, the transformations applied by JS code to external cookies are not necessarily limited to using the string type: for instance, an implementation of the base64 encoding would first convert the characters of the input string to integers, which are then used as indexes in a table that produces the characters of the output string. For these reasons, a system that aims to understand how external cookies are used needs to be able to gain detailed insights into the data flows of the JS code running on a page, *across all available data types* (and not limited to, e.g., just the string type).

Finally, given that we are interested in a large-scale measurement of external cookie usage across the web, and that a significant portion of JS code is likely to be minified/obfuscated to deter analysis [36], we need an automated way to analyze the results without resorting to manual analysis of all the encountered JS code. Ideally, such a technique should allow us to (i) distinguish cookies that are used for tracking purposes from those that are not related to tracking (e.g., timestamps), (ii) find the relationship between the domains setting the cookies and those that receive them, and (iii) categorize the common usage patterns of external cookies and filter out representative cases where manual analysis yields the most benefit.

3.2 Dynamic Taint Analysis

Our solution to the above challenges relies on information flow analysis techniques. Specifically, we leverage and extend Mystique [19], which implemented dynamic taint tracking for JS by modifying the Chromium browser's V8 JS engine and its Blink layout engine. Mystique's runtime taint propagation covers *all* JS data types, and it was originally built to identify browser extensions that leak the user's browsing history. We tailored Mystique to our usage scenarios. In the following, we document all the changes we made and show how the challenges mentioned in Section 3.1 are addressed.

3.2.1 Defining `document.cookie` *as Taint Source.* As discussed, external cookies are set by third-party JS code, that runs in the page, using the `document.cookie` property. For our analysis such cookies need to be distinguished from other first-party cookies (e.g., traditional first-party cookies, or cookies set by JS code that originates from the first party), so that we can properly taint and track them when they are read back (together with other first-party cookies). Thus, defining `document.cookie` as taint source needs to account for both read and write accesses. To that end, we made the following changes to Chromium: (i) whenever third-party JS code *writes* to `document.cookie` to either set a new cookie or update an existing one, we mark that cookie as tainted by recording it in a global table (keyed on *both* the cookie name and the domain on which it is set), and (ii) when `document.cookie` is *read* we taint the return value (i.e., a string that concatenates all the cookies of the website) if it contains cookies that we have previously marked as tainted. However, the value of `document.cookie`, which concatenates all the cookies of

the website, can potentially contain non-external first-party cookies. To avoid over-tainting those cookies, we extend Mystique's dynamic taint tracking implementation with precise byte-level taint tracking for the string type, which we discuss in the following section.

Incidentally, we consider scripts to be third-party when they originate from an eTLD+1 domain that is different from the first-party origin. We taint a cookie that is set via `document.cookie` when the *initiator* script is third-party. The eTLD+1 domain from which the initiator script is loaded is considered as the source domain of the cookie (we will use this definition in Section 4). The initiator script can be found from Chromium's call stack at the point when `document.cookie` is written to. We attribute the initiator script to be the bottom script on the call stack. Note that this attribution works even in the presence of asynchronously invoked JS code, since Chromium's call stack is able to trace across asynchronous functions [13]. For instance, if a script registered a callback which is later invoked asynchronously, the call stack will contain information about the original script responsible for registering the callback.

3.2.2 Byte-Level Taint Tracking for Strings. We implement byte-level taint tracking for strings by modifying V8 to associate a Boolean array with each string that is *partially* tainted: if byte `i` in the string is tainted, then offset `i` of the Boolean array would be set to `true`, otherwise `false`. Note that if a string is *fully* tainted, then it is not necessary to allocate a Boolean array for it - we treat tainted strings that do not have an associated Boolean array as fully tainted.

To keep this byte-level taint information up-to-date, we propagate the taint across string manipulation functions such as `concat` and `split`, as well as regular expression operations. This taint propagation is similar to Lekies et al. [27]. Our work, however, differs from theirs in the following ways: (i) Lekies et al. [27] only handled tainting for the string type, which is not adequate for measuring external cookies. As we mentioned, our approach requires tracking the complete data flow, *across all JS data types*, a capability we leverage from Mystique, and (ii) unlike Lekies et al. [27], precise byte-level taint propagation is not always possible with the approach taken by Mystique, on which our work is based (e.g., when taint is propagated across control-flow dependencies; see [19] for details).

3.2.3 Taint Sinks. To detect tainted values (i.e., information derived from external cookies) that are transmitted to third parties, we define the (i) `XMLHttpRequest`, (ii) `WebSocket`, and (iii) `src` attributes of HTML elements as taint sinks. Our system raises an alert whenever a tainted value reaches one of the sinks, for being sent to the network. Note that our approach considers all HTML elements that have a `src` attribute as taint sinks, as the browser would make a network request to fetch the resource pointed to by the `src` attribute. Since this results in a network request, affecting the `src` attribute of elements is a potential way for passing information to the third parties that provide the referenced resources.

3.2.4 Discussion. The modifications we made to Chromium address the first two challenges mentioned in Section 3.1. Specifically, as stated in Section 2, to the best of our knowledge, the DOM property `document.cookie` is the only interface that JavaScript has to the cookies of a website. Therefore, any external cookies that are set for a website will be detected by our modified Chromium browser; when these cookies are read back by JS code, they will be tainted by our instrumentation. Dynamic taint tracking then ensures that taint data is propagated as the JS code executes: any data that is derived from tainted values (e.g., base64-encoded) will also be tainted. Given this, we are able to track how external cookies are used by JS, and whether they are leaked over the network (i.e., whether the taint sinks are triggered by tainted values) back to third parties. Note that HTTP requests made to third-party domains do *not* have the external cookies embedded in the request header, as external cookies are set for the first-party domain. Thus, the only way for third parties to read back the external cookies is through JS that runs in the page, which ultimately needs to read them from `document.cookie`.

3.3 Detecting Tracking Cookies

Not all external cookies are used for web tracking. For example, some cookies may store a `true`/`false` flag to indicate whether a user has opted in to certain features of the website. Other examples include the client's language, geolocation, timezone, and timestamps. These values contain only coarse-grained information and as such they cannot uniquely identify a particular user. Note that in the case of timestamps, they are often paired with an additional random number to avoid collision and form a unique tracking ID (e.g., Google Analytics' `_ga` cookie [17]), but by themselves they are not uniquely identifying. This classification is in line with previous works [22].

To explore how external cookies are used for web tracking, we need to focus only on cookies that contain tracking IDs. To that end, we follow the methodology used in previous works [21, 22]. The key insight here is that tracking cookies have two important properties: (i) persistent value over time, and (ii) uniqueness across different browser instances. However, as the authors themselves noted in those works, their methods are intentionally conservative since they were interested in establishing lower bounds. Also, their methods are not specific to external cookies. Thus, we adapted their methods to effectively detect tracking IDs in external cookies. Specifically, we filter the external cookies recorded by our instrumented Chromium (see Section 3.2) according to the following criteria:

(i) The cookie should not be a session cookie, (session cookies are deleted by the browser when the session ends [16]).
(ii) The length of the cookie's unquoted value is at least 8 characters (unquote converts URL-encoded `%xx` sequences into their single character equivalents).
(iii) The cookie is always set when visiting the website using different browser instances.
(iv) The values of a cookie differ significantly across different visits to the same website using different browser instances.

Similarly to previous works, we define a "significant difference" in cookie values using the Ratcliff-Obershelp algorithm to compute similarity scores. Note that the above criteria are essentially the same ones used in previous works [21, 22]. Our methodology differs primarily in how similarity scores are computed. In [21, 22] the similarity scores are computed over the *entire* cookie values. However, we have frequently observed external cookie values such as "`{visitor_id: abcd, timestamp: 1234}`," where the common parts skew the similarity scores. In such cases, only the similarity between the actual visitor ID value is of interest. For this reason, we compute the similarity scores only after we have removed all the common parts between

the cookie values. Specifically, we do so by first removing all timestamps from the cookie values, and then all common subsequences having a length of more than 2. The latter is done by applying the longest common subsequence (LCS) algorithm and removing the LCS from both values, and repeating until the LCS length is less than or equal to 2. The similarity score of the residual values are then computed using Ratcliff-Obershelp, and we set a cutoff value of 66% (i.e., the original cookie values are significantly different when this score is below the cutoff). The same cutoff value of 66% was also chosen in Englehardt et al. [21].

3.4 Heuristic-Based Identifier Matching

Having identified external cookies that contain tracking IDs, we are then interested in measuring (i) which of these tracking cookies are later read back by third-party JS and leaked over the network, and (ii) the relationship among the entity (i.e., TLD+1 domain) that originally set the cookie, the entity that triggered the reading back and leaking of this cookie, and the entity that receives the leaked cookie. However, one technical challenge here is that since Mystique [19], on which we base our dynamic taint tracking implementation, only tracks the binary status (i.e., tainted or not) of JS values, it does not provide information at taint sinks about the provenance of the tainted values (i.e., the tracking cookies from which the tainted values triggering the sinks are derived). Thus, we need an orthogonal method to automatically extract this information. In order to address this challenge, we design and apply a set of heuristics that attempt to match the tainted values that triggered a taint sink against the identified tracking cookies. For convenience, we hereafter refer to these values (i.e., values that triggered a taint sink) as *sink objects*.

It should be noted that these heuristics are complementary to our taint tracking mechanism: they allow us to (i) verify the results of taint tracking by confirming that indeed the cases of triggered sink objects correspond to true positives, and (ii) filter out cases of information flows that do not include unique identifiers and thus do not affect the user's privacy (as we mentioned in Section 3.3, some tracking cookies contain both unique tracking ID and non-unique parts). If we did *not* have the taint tracking mechanism, and we relied solely on heuristics for matching the cookies against the network traffic, then (i) we would not have any information about which third-party scripts set each external cookie (nor whether the external cookies contain tracking IDs), and which read and diffuse these cookies over the network; indeed, it is not even possible to know which of the first party cookies are external cookies, and (ii) we might miss cases where the leaked information is transformed (i.e, encrypted or obfuscated) before being sent.

The first and simplest heuristic tries to match the name or value of a tainted cookie with a substring in the sink object. In the case where the sink object is a URL, we extract the URL parameters (e.g., the argument passed to `XMLHttpRequest`'s `open` method), and check whether the name or value of a tainted cookie matches the extracted URL parameters. In the great majority of cases, this heuristic was able to match the *entire* name or value of the tainted cookie against the sink object. However, there were also some cases where the name or value of the cookie did not match a substring of the tainted sink object, but it was evident that either the name or the value of the cookie was generated by the concatenation of two or more string literals (e.g., `value1.value2`, `value1_value2`). To also account for such cases, we apply a slightly more complex heuristic that splits the cookie value to substrings, based on a set of special characters, and attempts to match these substrings with parts of the sink object.

Finally, we apply a heuristic that tries to detect cases that involve transformed values. This heuristic, similarly to the previous one, splits the tainted cookie values into substrings and then applies the base64 encoding, and the MD5, SHA1, and SHA256 hashing algorithms both to the entire cookie value and the substrings, and attempts to match them with parts of the triggered tainted sink object. While this heuristic is more complex than the previous ones, it allows us to verify cases that we cannot verify manually, where the tainted information is transformed before reaching the sink object. We present in Section 4 the results of applying our heuristics to the data gathered from crawling the Alexa top 10K websites.

3.5 Experimental Setup

We measure the prevalence of external cookie usage by crawling the Alexa top 10K websites with our instrumented Chromium browser. Each instance of Chromium runs in a separate Docker [5] container. After each website has finished loading, we wait for an additional two minutes, to allow sufficient time for the JavaScript code on the page to execute. This process is then repeated two more times inside the same Chromium instance, since external cookies are typically set on the first visit and are read back and sent only on subsequent visits. Note that depending on the purpose of the analysis, we either launch a fresh instance of Chromium (i.e., no prior state) for *each* one of the websites we analyze, or a fresh instance *per cluster* of websites (e.g., websites that have external cookies with the same name) and visit each website of a cluster in the same Chromium instance *one by one*. We will provide more details about this type of analysis in Section 4.

Finally, we parallelize the analysis by running the Dockerized instances of the Chromium browser on a local Kubernetes cluster [6]. The analysis tasks (e.g., the URLs to visit) are distributed to workers via Redis [7]. We note that since the tracking cookie detection algorithm described in Section 3.3 requires two separate browser instances visiting the same website at about the same time, we enqueue each website twice consecutively to Redis, so that they are picked up, one immediately after the other, by the workers in the Kubernetes cluster. Since each website is visited by two separate browser instances, essentially this means that there are two separate crawls of the Alexa top 10K. Hereafter we refer to these two separate crawls as the *main crawl* and the *control crawl*, and we treat the website visit that results from the first time it is enqueued as belonging to the main crawl (and thus the visit from the second enqueue as the control crawl).

4 TAINT ANALYSIS RESULTS

In this section, we report the results of our analysis on the Alexa top 10K websites. We begin by describing the raw data from our crawl and giving an overview on the number of external cookies that we encountered. This is followed by a taxonomy breakdown of the recorded external cookies, based on the output of our algorithm for the detection of tracking cookies that is described in Section 3.3. We then present the results of our heuristics (Section 3.4) and explore in detail the relationship between the *source domains* (i.e., entities that originally set external cookies) and the *sink domains* (i.e., entities that receive information derived from confirmed tracking cookies).

4.1 Overview of the Initial Crawl

In total, we recorded external cookies being set in 9,772 (97.72%) out of the Alexa top 10K websites that we crawled. Table 1 gives an overview on the number of external cookies that were recorded in our crawl. Recall from Section 3.5 that in our experimental setup we essentially have two simultaneous crawls, and we refer to them as the *main* and the *control* crawl, respectively. We show in Table 1 the statistics for both crawls. We note that in both crawls each website is visited by a fresh instance of Chromium.

To give a more comprehensive picture regarding the external cookies on the Alexa top 10K websites, we count the number of external cookies using a variety of different criteria. In Table 1 we present both the number of *unique* cookies and the number of *instances* of such cookies (i.e., the total number of times those unique cookies were set/updated via `document.cookie`), according to different criteria. For example, if we define unique cookies based on the tuple `<cookie_domain, js_domain, cookie_name>`, where `js_domain` denotes the third-party domain serving the JavaScript code that sets the cookie, then in our main crawl we observed 100,146 unique external cookies, and our Chromium instrumentation recorded that these cookies were updated (via `document.cookie`) for a total of 185,910 times. From the numbers in Table 1 one can make a few observations: since the number of unique cookies according to `<js_domain, cookie_name>` is much lower than those based on `<cookie_domain, cookie_name>`, it follows that some third-party domains are responsible for serving scripts that set external cookies in a large number of websites. Another similar example is the criterion `<js_domain, cookie_name>` vs. `<cookie_name>`, which indicates either cookie name conflicts, or that the same cookie-setting script is hosted on multiple domains (e.g., to avoid filter list blocking [20]).

In Table 1 we also show the subset of cookies in each category that correspond to *non*-session cookies (i.e., numbers in parentheses). These numbers are provided to give a sense of the input to our tracking ID detection algorithm described in Section 3.3, the results of which we discuss next. Unless otherwise specified, hereafter we count unique external cookies based on the tuple `<js_domain, cookie_name>`, since it is the most appropriate given the scope of this paper, and allows us to easily cluster different first-party websites based on the external cookies that are set on them.

4.2 Taxonomy of Non-Session External Cookies

Next, we present the results of our tracking ID detection algorithm, that we described in Section 3.3. Recall that the algorithm looks for tracking IDs in *non*-session cookies, and as shown in Table 1, in the main crawl there are 13,323 non-session cookies (by `<js_domain, cookie_name>`). Note that since session cookies are deleted when the current session ends [16], by themselves they are not suitable for persistent tracking purposes. Therefore, in the rest of this paper we primarily focus on non-session cookies.

Table 2 presents a taxonomy of the non-session cookies collected from our main crawl. The taxonomy categories are based on the various conditions that need to be satisfied in order for a cookie to be considered by our algorithm as a tracking cookie. We give the details of each category below.

4.2.1 Tracking ID. This category includes the non-session external cookies that our algorithm identifies as tracking cookies. Out of the 13,323 unique non-session external cookies, we found 4,212 (31.61%) that contain tracking IDs. We extracted the URLs serving the scripts that set these cookies and cross-referenced them against the crowd-sourced filter list EasyList/EasyPrivacy (on which the popular Adblock Plus [10] extension is based). Alarmingly, we found that 1,673 (nearly 40% of UID-containing cookies) would have evaded filter list blocking. We will present more details about these tracking ID cookies in Section 4.3, where we explore (i) the relationship between domains serving the scripts that initially set those cookies (i.e., *source domains*), and domains serving scripts which later read back these cookies and send them over the network (i.e., *retriever domains*), and (ii) the top 10 source domains, which set the most unique tracking ID cookies, and the top 10 sink domains that received the most unique tracking ID cookies. Note that for a cookie that is read back and sent over the network, the retriever domain does not necessarily need to be the same as the sink domain. For example, a third-party script might collect all external cookies set by other well-known trackers (e.g., Google Analytics) and send this information back to their servers, a commonly encountered scenario which we discuss in Section 4.3. In that section we will also give sample values of the top 10 tracking ID cookies of this category that are set on the most websites in our crawl of the Alexa top 10K, shown in Table 3.

The rest of the categories in this taxonomy of the non-session cookies are regarded by our algorithm as not containing tracking IDs, so they are not likely to be used for persistent tracking of users. For completeness, we describe each category, with the purpose of giving a sense of what the values of these cookies contain, as well as showing which of the filtering conditions in our tracking ID detection algorithm they failed to satisfy. We provide sample values of cookies in each category as appropriate.

4.2.2 Similar Value. One of the more interesting categories in our taxonomy is the category of cookies that otherwise passed all the conditions of the tracking ID detection algorithm (i.e., they are non-session cookies, having minimum unquoted value length, and non-constant values between visits by different browser instances), except their values do not differ significantly in the two crawls. That is, the Ratcliff-Obershelp similarity score of their values between the two separate crawls are above our cutoff of 66%, after all the timestamps and common parts are removed. As mentioned in Section 3.3, timestamps *by themselves* are not uniquely identifying. We implemented our tracking ID detection algorithm so that for a given cookie under consideration, it keeps track of whether timestamps are removed from its value as part of the detection process. In total, *all but one* cookie in this category have their values differ between the two crawls due to having different timestamps. Examples of this category of cookies include `_ym_d` set by scripts from yandex.ru (e.g., on academic.ru) whose value is composed entirely of a timestamp; another example is `smct_last_ov` by smct.io (set on e.g., theclutcher.com), with a value that is a serialized JSON object, e.g., `[{"id":35060, "loaded":1594819221556, "open":null, "eng":null, "closed": null}]`. The latter example highlights the importance of removing *both* the timestamp and common parts from the cookie value, in the way that our algorithm does, prior to computing the similarity score, since the properties in a serialized JSON object are not guaranteed to have the same order [14].

Incidentally, the single cookie in this category whose value difference between the crawls is not due to timestamps is `ckBAHAADS`, that

Uniqueness Criterion (Tuple)	Main Crawl		Control Crawl	
	# Instance	# Unique	# Instance	# Unique
<cookie_domain, js_domain, cookie_name>	185,910 (112,185)	100,146 (65,871)	184,619 (111,108)	99,725 (65,273)
<cookie_domain, cookie_name>	184,714 (112,024)	98,746 (65,250)	183,454 (110,944)	98,335 (64,648)
<js_domain, cookie_name>	138,565 (93,300)	26,632 (13,323)	138,026 (92,380)	26,560 (12,906)
<cookie_name>	136,585 (92,660)	23,909 (11,876)	136,093 (91,776)	23,869 (11,543)

Table 1: Statistics on the number of external cookies, both in the main crawl and the control crawl of the Alexa top 10K. The numbers in parentheses indicate the subset in the same category that are non-session cookies.

Category	#Instance (% Total)	# Unique (% Total)
Tracking ID	76,617 (82.12%)	4,212 (31.61%)
Similar Value	6,500 (6.97%)	1,451 (10.89%)
Constant Value	2,245 (2.41%)	1,078 (8.09%)
Short Value	6,838 (7.33%)	5,690 (42.71%)
No Control Value	1,100 (1.18%)	892 (6.70%)
Total	93,300 (100.00%)	13,323 (100.00%)

Table 2: Taxonomy of non-session external cookies collected from the main crawl.

is set by bahamut.com.tw on the website gamer.com.tw, and its values are `{"FA": {"a3": 9, "a1": 1}}` and `{"FA": {"a3": 11, "a1": 0}}` in the main and control crawl, respectively.

4.2.3 Constant Value. This category includes the non-session external cookies that are set both in the main and the control crawl, but their values are always the same, despite the fact that the websites were visited by different browser instances, and thus are not unique per user. Many of these cookies would have fallen under the "Short Value" category if we used a larger cutoff for the minimum value length (e.g., the JavaScript literal `undefined`). We also observe many cookies that contain information specific to the visited website, e.g., the website's domain name.

In addition, we also see a few cases where the cookie values resemble a user ID, for instance the `ucfunnel_uid`[1] cookie, set by aralego.net, when visiting pcstore.com.tw. Since all of our browser instances in the crawl run from the same configuration, this might indicate the use of fingerprinting. Indeed, one limitation of our tracking ID detection algorithm is that it is unable to find cases where the cookie values are always the same due to them being generated from fingerprinting the browser. In Section 4.4, we explore more systematically how many such cases there are in the external cookies collected from crawling the Alexa top 10K.

Nevertheless, here we attempt a first approximation to addressing the above limitation, by cross-referencing with the popular Disconnect list [15], which is used by Firefox's Enhanced Tracking Protection to block known fingerprinters. Overall, we found 8 unique cookies from this taxonomy category that are set by known fingerprinters, according to Disconnect. However, we manually verified that almost none of these 8 cookies contain unique user IDs, except one that appears to be a long UID-containing string.[2] Note that the `ucfunnel_uid` cookie, mentioned above, was not among the 8 identified by Disconnect. On the other hand, as a reference we remark that Disconnect identified 36 from the "Tracking ID" category as being set by known fingerprinters (more precisely, by scripts served from known fingerprinter domains).

4.2.4 Short Value and No Control Value. Cookies in the "Short Value" category are only set with values that fail to meet the minimum value length requirement of our tracking ID detection algorithm (i.e., their unquoted value length is less than 8). Example values for cookies frequently found in this category include boolean flags (e.g., `0`, `1`, `true` and `false`), JavaScript literals such as `null`, as well as short strings such as "en-US", "enabled", etc. In this paper, as in previous works [21, 22], we do not consider these to be UID-containing.

Finally, the "No Control Value" category contains cookies that were observed in the main crawl, but were not set in the control crawl. These cookies comprise 6.70% (totaling 892) of all the non-session external cookies in the main crawl. We do not consider such cookies to be stable enough to reliably track visitors to a website.

4.3 Cross-Domain Cookie Sharing

Having established which of the external cookies recorded in the main crawl can be used to persistently track users, we now focus on the relationships among: (i) the *source domains* that serve scripts responsible for initially setting the tracking ID cookies, which we identified in Section 4.2.1, (ii) the *retriever domains* which serve scripts that read back the tracking ID cookies and send them over the network, and (iii) the *sink domains* which receive information derived from tracking ID cookies, sent to them by scripts that are served from the retriever domains. In particular, we are interested in cases where the sink domain is different from *both* the source domain, and the cookie domain (i.e., domain on which the cookie was set), since this indicates cross-origin sharing of identifiers, which further undermines user privacy.

As mentioned our taint tracking implementation only tracks the binary status (i.e., tainted or not) during taint propagation. We overcome this limitation by proposing an orthogonal method (see Section 3.4), that uses heuristics to match sink objects against the identified tracking ID cookies. In the following we give an overview of the results from our heuristic matching, and present the domain relationships that we identified.

4.3.1 Overview of Heuristic Matching Results. We applied our heuristic matching algorithms described in Section 3.4 to the 4,212 cookies that we identified to be containing tracking IDs (see Section 4.2.1), and we were able to match 3,256 (77.30% of all identified UID-containing cookies) against at least one sink object that was recorded

[1] Having values of the form `88d501e0-40f3-3fcf-bf48-c8fa59bc7efd`.

[2] Cookie: `segmento`. It is set on oi.com.br by a script that originates from maxmind.com. Its value in our crawl is (truncated): `56c4339f58ee7410VgnVCM10000031d02 00a____-e99eecf7c7cc5410VgnVCM10000031d02...`

Cookies' Source Domain	Websites	Top External Cookie Shared to Third Parties			
		Cookie Name	Websites	Third-Party Destinations	Example Value
google-analytics.com	3,620	_ga	3,456	329	GA1.2.1687927199.1594842303
facebook.net	2,377	_fbp	2,377	76	fb.1.1594781590601.135769710
googletagmanager.com	1,124	_gcl_au	882	124	1.1.2086254180.1594824717
doubleclick.net	792	__gads	742	114	ID=18d6b32f18c77049-22c915ffccb70097: T=1594800595:S=ALNI_MZEEA_JlMOtwGQyjjw4rJp4cwDL2A
googlesyndication.com	551	__gads	551	33	ID=7b7fb7d54109a6c6:T=1594819953: S=ALNI_MYzCXjBjUDNFPiMFgf55XXCoZAVBg
cloudfront.net	469	__asc	310	4	678afacf1735170ad8631aabda0
go-mpulse.net	280	RT	279	424	"si=a1ed29ce-7795-47c5-9830-5745fa3c04bd& z=1&dm=oracle.com&ss=kcnp67pf&sl=0&tt=0..."[3]
chartbeat.com	243	_cb	243	6	DQVCY6B3K-HtDvrvFv
cookielaw.org	242	OptanonConsent	242	381	consentId=5b39d6ae-72ba-47d9-9250-4df42b23f5d6& isIABGlobal=false&interactionCount=0...[3]
adobedtm.com	239	mbox	153	15	session#50141a1754164b66900a3a2f030f03cd #1594825706|PC#50141a1754164b66900a3...[3]

Table 3: Top 10 source domains, whose cookies are then shared to other third parties. Note that we do not include cases where the destination is the same as the source domain. Also, the third-party destinations column represents the total number of third parties that receive the corresponding external cookie, across all websites.

Destination Domains	Source Domains	Cookies
google-analytics.com	198 (187)	427 (405)
doubleclick.net	189 (182)	432 (380)
facebook.com	121 (118)	224 (212)
omtrdc.net	80 (45)	325 (93)
criteo.com	59 (44)	73 (56)
adnxs.com	51 (30)	61 (34)
openx.net	51 (23)	67 (26)
googleadservices.com	48 (46)	75 (73)
taboola.com	45 (43)	70 (68)
bing.com	44 (43)	68 (67)

Table 4: Top 10 third-party destination domains that receive external cookies by other third parties (different source domain).

during the crawl by our taint tracking system. The main reason why our heuristics were not able to match all the UID-containing cookies to the flagged sink objects is due to the complex transformations that take place before a cookie value is eventually leaked over the network. Indeed, that is why such a measurement study cannot completely resort on matching heuristics, but also needs to use a taint tracking system for being able to track the transformations.

We manually inspected several randomly chosen cases of the matched cookies, and we were able to verify that all of these indeed correspond to true positive cases that were correctly detected by our system. Through the manual inspection process we identified which scripts set the particular cookies (and verified that those are the same as reported by our system), and we were able to manually track how those cookies are transformed, and how they reach the sinks. Based on our manual analysis we are confident that the cases reported by our methodology are correct in the sense that they do all correspond to true positive cases. However, unfortunately we are not able to draw any conclusions regarding any possible false negatives (i.e., cases of such cookies that are leaked but our system fails to detect) as this would require us to manually follow all the cookies of a website, which is an almost impossible task to perform manually.

4.3.2 Cross-Domain Cookie Sharing. Next, we focus on exploring cases where the external cookies by one third party are shared to multiple parties (i.e., third-party destinations) different from *both* the source domain (i.e., the third party that initially set the cookie), *and* the domain on which the cookie was set. We refer to this case as *cross-domain cookie sharing*. That is, in the following we focus on flows of cookie information from one third-party to another, like the example case that is presented in Figure 2 and thus, we exclude cases where the cookies are sent back to the same third party that have set them. Note that this also excludes the scenario where scripts loaded from a CDN controlled by the first party sets the cookies which are then only sent back to the first party.

Out of the 3,256 cookies that were matched by our heuristics, 2,354 (55.89% of all UID-containing cookies) are leaked to a third party that is different from both the source domain and the domain on which the cookie was set. For convenience, we will refer to this set of cookies as *shared cookies*. Table 3 and Table 4 show the highlight of our findings. In Table 3 we focus on the *source* of shared cookies, and we list the top 10 third parties that have their external cookies shared to other parties (i.e., destinations). We rank the source domains according to the number of first-party websites in which we observe one of their cookies being shared or leaked to other third parties, and we provide information about their top cookie that is shared in most cases. For example, we find 3,620 websites that have sink objects sharing external cookies by google-analytics.com to other third-party domains, with the top shared cookie being _ga, which is shared in 3,456 of the websites. The most interesting finding, however, is that this cookie is shared to 329 different third-party domains in total, most of which clearly do not belong to Google. Unlike traditional third-party HTTP cookies, where cookie synchronization is typically performed between two

[3]We truncated these cookies' values due to space limitations.

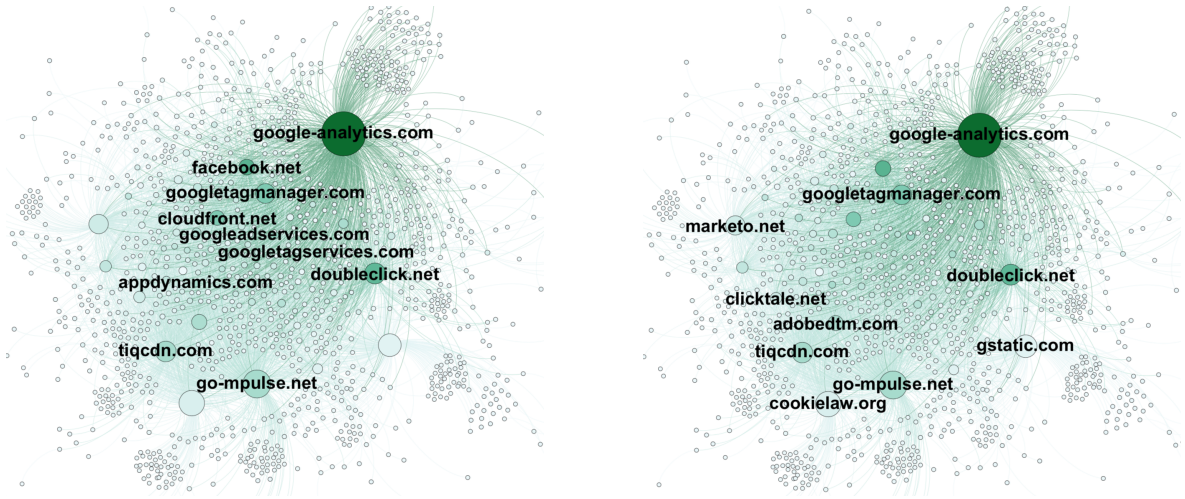

Figure 3: Directed graph representing the unique connections between source domains and third-party destination domains. The size of the nodes is proportional to their overall degree (i.e., number of connections). The color of nodes represents their number of unique incoming (left) and outgoing (right) connections, where the third-party domains act as a destination/source respectively.

third parties (with clear consent between the parties involved, i.e., through URL redirection), our findings show that external cookies are facilitating the extensive sharing (or leakage) of information to multiple third parties.

In Table 4 we focus on the sink domains that receive external cookies set by other parties, and we list the top 10 ranked by the number of different source domains whose external cookies they received. For each domain in Table 4 we also show the total number of unique cookies received. For instance, we find that and google-analytics.com and doubleclick.net receive a total of 427 and 432 unique UID-containing cookies (that were matched by the heuristics), which were set in total by 198 and 189 different source domains, respectively. These numbers demonstrate how extensive the utilization of external cookies is for information sharing currently on the web.

In addition to the above, we also identify the retriever domains (i.e., domains serving the code that is responsible for reading the cookies and diffusing them). If the retriever domain is the same as the source domain of the cookie, this indicates that the source third party cooperates with the destination third party and willingly shares the cookie. Otherwise, if the source and retriever domains are not the same, it indicates that the destination third party is "stealing" the cookie that was set by another party, possibly without its consent (i.e., this is a special case of cross-domain cookie sharing, in which the source domain did not initiate the sharing). We show the corresponding numbers in parentheses in Table 4. For example, our analysis reveals that google-analytics.com, which receives cookies set by 198 different parties, uses its own code to read and send the cookies of 187 of them. Similarly, the scripts loaded from doubleclick.net are responsible for triggering the sinks that send the cookies of 182 parties back to itself.

In total, we observe cross-domain cookie sharing in 5,635 out of the 9,772 (57.66%) websites that have external cookies. In particular, we detect 718 source third-party domains that have their external cookies shared, and 1,778 third-party destination domains that receive these shared cookies (correspondingly, for the "stolen cookies", i.e., a special case of cross-domain sharing, they are observed in 4,735 websites, or 48.45% of all websites that have external cookies; these stolen cookies are set by a total of 546 source domains, and leaked to 1,397 sink domains). Figure 3 shows the overall relationships among the domains engaged in cookie sharing, by representing the graphs of the incoming and outgoing connections between the various third parties. Interestingly, as can be seen in Figure 3, apart from a few Google-owned services that have both a high in-degree and out-degree (i.e., google-analytics.com, doubleclick.net), the other third parties have either a high in-degree (incoming connections from other parties, i.e., receiving external cookies' values) or a high out-degree (i.e., their cookies are shared to multiple destinations). Finally, in Figure 4 we present the distribution of the number of shared and stolen cookies per website, and in Figure 5 we plot the distribution of sink domains per website that correspond to the shared and stolen cookies.

4.3.3 Case Study: Google Analytics. We close this section by giving a deeper insight into the results presented previously in Section 4.3.2. Specifically, we manually examine cases regarding Google Analytics (GA), which as shown in Tables 3 and 4 (and Figure 3) represents the party whose external cookies are most frequently stolen, as well as being the party that receives the most external cookies set by other domains. Considering that Google owns several ad/tracking-related domains, such as `doubleclick.net`, `googletagservices.com`, etc, we further filter from our cross-domain sharing cases that involve `google-analytics.com`, but with a counterparty (i.e., receiving or stolen-from domain) that is clearly not Google property. We also focus our attention on the `_ga` cookie, which is a well-known cookie from Google Analytics that contains a per-site client ID [11].

Note that while in the following we are able to confirm a third-party script *actively* reading from the `_ga` cookie and sending its value back to a third-party server, we do not know the purpose of such leaks, or whether this exchange of information is consented to by both parties, since we do not have visibility into the backend processing logic. Nevertheless, our analysis shows that external cookies do not only

allow third parties to bypass the restrictions imposed on third-party HTTP cookies, but provide an easy way for third parties to exchange information *en masse* and cooperate, which endangers users' privacy beyond the extent of traditional third-party cookies: consider for example a third-party service that sets an external first-party cookie on a website that the user visits. That cookie would then be accessible to all other scripts, third-party or otherwise, executing in the first-party context. In that sense, all third parties that have their scripts in the page can access information provided by all other third parties, even without an explicit cooperation agreement between those third parties (as contrasted with traditional cookie-sharing agreements). We demonstrate this point more concretely below.

Adobe Demdex: We first focus on a case where external cookies set by Google Analytics are leaked to other third parties. We observed scripts from `adobedtm.com` sending the `_ga` cookie to `demdex.net`, which is a domain controlled by Adobe. The value of the `_ga` cookie is sent via `XMLHttpRequest` embedded in the body of a POST request (with the body being a URL-parameter-style string and the parameter name being `c_gacid` in this string, along with other parameters; the leading `GAx.x.` prefix is stripped from the value of the `_ga` cookie before it is sent, see Table 3 for an sample value of this cookie). An example of this leakage can be found on `http://www.uplus.co.kr` at the time of this writing. In our dataset, we found this cross-domain leakage of the `_ga` cookie to `demdex.net` in 17 of the top 10K websites. We manually analyzed the JS code responsible for sending the cookies, and found that the script, which is obfuscated, reads the `_ga` cookie at multiple places in its source code. Although to the best our knowledge we do not know whether Adobe and Google Analytics have tracking ID exchange agreements, this case illustrates the ease with which third parties can steal information from each other, without the consent of the affected parties.

GA Custom Dimensions: Here we focus on the case where external cookies set by other third parties are leaked to Google Analytics. In Table 4, we have shown that Google Analytics also receives external cookies that were set by 198 other third-party domains, and that in total Google Analytics uses its own code (that is, code served from `google-analytics.com`) to read cookies from 187 of them. We manually examined their corresponding sink objects, and found that the leaked external cookies containing unique tracking IDs, which were set by scripts served from domains that Google does *not* control, were being sent to `google-analytics.com` encoded in the HTTP request's URL as one of its `cdX` parameters, where `X` is a number. Upon further investigation, we found that this is due to a Google Analytics feature known as "custom dimensions and metrics" [12], which can be enabled and configured by the website owners, and that the `cdX` parameters encode additional metrics (in this case external cookies set by other third parties) that Google Analytics does not automatically report by default. We point out that this usage scenario highlights the abusive nature of external cookies: website owners, instead of using their own infrastructure to record external tracking cookies set on their websites, abuse the custom dimensions feature to aggregate all external tracking cookies to another third party tracking provider, without the knowledge of the users and in doing so possibly also violating the terms of agreement with the third parties whose external cookies are reported to Google Analytics.

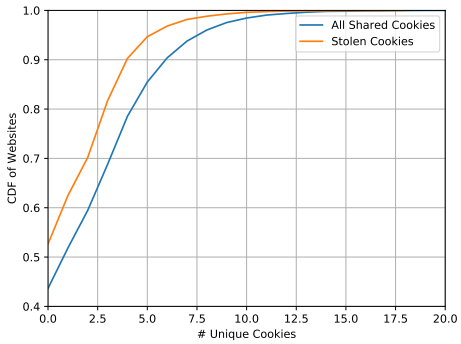

Figure 4: CDF of the number of both shared and stolen cookies per website of the Alexa top 10K.

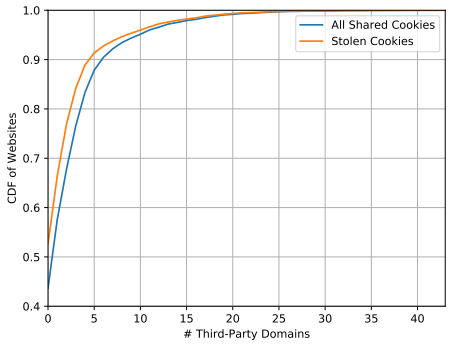

Figure 5: CDF of the number of sink domains per website corresponding to both the shared and stolen cookies encountered on each website.

4.4 Fingerprinting

Finally, we address the limitation of our tracking ID detection algorithm at identifying tracking ID cookies whose values are always the same due to them being generated based on fingerprinting. As mentioned in Section 3.3, this limitation stems from the fact that the ID detection algorithm relies on comparing the difference in values set for the same cookie on separate crawls. In Section 4.2.3, we manually found a cookie `ucfunnel_uid` whose value resembles a UID string (indeed the cookie's name suggests it is used as an ID cookie), and remains constant on our two separate crawls. In this section, we explore such cases in a more systematic manner, and attempt to quantify how many potentially UID-containing cookies that our ID detection algorithm missed. We remark that our purpose in this section is to establish that the results we report in this paper is not significantly impacted by the limitation of our ID detection algorithm, and we leave to future work the automatic identification of fingerprinting-generated UID cookies.

Our strategy to detecting fingerprinting-generated cookies involves: (i) we cluster the Alexa top 10K websites that we visited in the main crawl (when we visited each website using a fresh instance of Chromium) based on the names of the *non-session* cookies that are

Script Domain	Cookie Name	# Websites	Value
aralego.net	ucfunnel_uid	2	88d501e0-40f3-3fcf-bf48-c8fa59bc7efd
clarip.com	c_uuid	5	2501186645373654028409053736260080024
futurecdn.net	FTR_FingerPrint	4	6b2aad5d1452f2e65a0c1874aa09fee0

Table 5: Manually identified fingerprinting-generated cookies.

set on them, so that websites that have a cookie with the same name are placed in one cluster, and (ii) we set up our crawling infrastructure (see Section 3.5) to crawl the websites in the same cluster in a sequential order, one after another, using the same browser instance (so the browser state is kept in between visits to different websites). We disable third-party cookies in this crawl to prevent trackers from using them to synchronize their UIDs across first-party boundaries (such that if any cookies are set with the same UID, then it is highly likely that the UID is generated from fingerprinting). Since the tracking ID detection algorithm does not work here, we do not need to collect control values for the cookies, and thus the websites in each cluster are visited only once and there is no distinction of main crawl and control crawl, as we had before. Note that the total number of clusters is the total number of cookie names that are common across websites (i.e., websites that have more than one cookie name in common will appear in more than one cluster). In our main crawl, there are 919 non-session unique *cookie names* that appear in more than one website, so in total we have 919 clusters for this crawl.

We find potentially fingerprinting-generated UIDs in the crawl result by looking for cookies whose value remains the same across websites in the same cluster. In total, we found 166. However, this does not indicate that all of these cookies contain tracking IDs: 119 of them do not meet the value length requirement of our tracking ID detection algorithm. We manually perused the values of the rest of the 47 cookies: the majority do not contain unique identifying information (e.g., only the geolocation, IP address, timestamps, and short strings similar to what we reported in Section 4.2.4). Nevertheless, we list in Table 5 the 3 cookies (including the previously mentioned `ucfunnel_uid`) that we found to be highly indicative of fingerprinting.

5 RELATED WORK

Our work presented in this paper is related to previous research on the topics of (i) measurement of, and defense against, third-party web tracking, and (ii) JavaScript sandboxing.

Third-party tracking. The web's power comes from its ability to link to third-party contents, but this also enables third-party tracking. Krishnamurthy et al. [25] examined the prevalence of third-party tracking by carrying out a longitudinal study. Mayer et al. [29] proposed a web measurement platform, FourthParty, to survey the policy and technology issues involved in third-party tracking. Englehardt et al. [21] conducted a large-scale measurement of Alexa top one million websites. Other than pure measurements, previous work also proposed defenses against third-party tracking [32, 34]. Pan et al. [32] proposed an anti-tracking browser that isolates unique identifiers into different browser principals so that the accuracy of those identifiers is significantly reduced. Roesner et al. [34] explored various techniques employed by trackers. Although the authors acknowledged that external cookies can be used to track repeat visitors to the same website (i.e., case of web analytics), they underestimated their potential in bypassing third-party cookie blocking and facilitating cross-domain exchange of tracking IDs, a capability which we examine at length in this paper. Franken et al. [24] evaluates third-party cookie policies on the current web. A recent work, by Fouad et al. [23] explored different techniques employed by trackers, and among them found that the values of first-party cookies can be leaked to third parties. Our work adds another perspective to this ongoing line of research by focusing on first-party cookies that are set by third-party code, and explore how they relate to third-party tracking on the web.

Lastly, Sanchez-Rola et al. [35] is a closely related work that is concurrent to ours, which studies "ghost cookies" that is conceptually the same as external cookies. In that work the authors propose the notion of cookie trees to systematically explore the relationships among entities engaged in web tracking, but they lack the detailed JavaScript data flow information that is afforded by an analysis system such as Mystique, which we leverage in this work. Combining the strengths of their methodology and ours is a promising direction for further bettering the understanding of web tracking.

JavaScript sandboxing: The abuses of first-party cookies set by third-party JS code, which we focus on in this paper, can be eliminated or otherwise mitigated, if browsers implement sandboxing for scripts loaded from different origins. The Chrome browser already enforces a form of sandboxing where the content scripts injected by browser extensions cannot access any variables or functions created by the normal JavaScript code running on the page, or by other content scripts [4]. Previous research in sandboxing JavaScript by Agten et al. [18] proposes an approach to securely integrate third-party JavaScript code that achieves complete mediation. They improved on existing works that rely on instrumenting untrusted code on the server side to a safe subset of JavaScript (e.g., [33]), or implementing a reference monitor inside the browser (e.g., [30, 37]). Adapting these research efforts to mitigate the privacy impact of first-party cookie tracking is a direction for future work.

6 LIMITATIONS AND FUTURE WORK

The methodology we proposed in this paper, with regard to measuring web tracking, can be further improved in several ways. We briefly summarize in this section the directions in which future research can build on our work. These include: (i) taint engine improvements, specifically the capability to assign "colors" to external cookies set by different origins (i.e., multi-color taint), which as we mentioned, would improve the precision of the detection results since this obviates the need to rely on heuristic matching of sink objects against the recorded external cookies, and allows one to single out more sophisticated cases that encode tainted values in ways not anticipated by the heuristics, (ii) considering other persistent storage mechanisms provided by the browser, such as `localStorage`, as taint source, to see whether they are being utilized similarly for web tracking, and finally, (iii) orthogonal to the work presented in this paper, instead of relying on matching

the eTLD+1 domain to define third-party domains (and thus third–party JS code), our results could further benefit from having a systematic method to map a domain to its owner organization (e.g., both `google-analytics.com` and `doubleclick.net` belong to the same organization, which is Google), since this would reflect more closely the relationships among the entities in the web tracking ecosystem.

7 CONCLUSION

In this paper, we examine a web tracking method that abuses first-party cookies that are set by third-party JavaScript code, which has been largely neglected by previous works. We refer to these third-party-originated first-party cookies as external cookies, and they are used for circumventing browser policies that seek to block traditional third-party cookies as well as facilitating mass exchange of tracking IDs. In order to measure how external cookies are being used for web tracking, we implemented dynamic taint analysis for cookies in the Chromium browser, and use it to analyze the Alexa top 10K websites. We show that external cookies are already widely used: they are encountered on 9,772 (97.72%) of the Alexa top 10K, and that 57.66% of these websites have third parties that exchange tracking IDs stored in external cookies. Our results clearly indicate that existing mitigation techniques, such as blocking third-party cookies and using filter lists such as EasyList/EasyPrivacy, are not adequate to protect users from third-party tracking, and thus the web needs additional countermeasures to protect users.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their insightful feedback and comments. This work was supported by the National Science Foundation (NSF) under grant CNS-1703375.

REFERENCES

[1] PCMag - Firefox 22 to Disable Third-Party Cookies by Default. `https://www.pcmag.com/news/308420/firefox-22-to-disable-third-party-cookies-by-default`, 2013.
[2] Mozilla - Firefox 63 Lets Users Block Tracking Cookies. `https://blog.mozilla.org/security/2018/10/23/firefox-63-lets-users-block-tracking-cookies/`, 2018.
[3] Chrome - chrome.cookies. `https://developer.chrome.com/extensions/cookies`, 2019.
[4] Chrome - Content Scripts. `https://developer.chrome.com/extensions/content_scripts`, 2019.
[5] Docker. `https://www.docker.com/`, 2019.
[6] Kubernetes - Production-Grade Container Orchestration. `https://kubernetes.io/`, 2019.
[7] Redis. `https://redis.io/`, 2019.
[8] Web Technology Surveys - Usage of JavaScript for websites. `https://w3techs.com/technologies/details/cp-javascript/all/all`, 2019.
[9] WebKit - Intelligent Tracking Prevention 2.2. `https://webkit.org/blog/8828/intelligent-tracking-prevention-2-2/`, 2019.
[10] Adblock Plus. `https://adblockplus.org/`, 2020.
[11] Cookies and user identification. `https://developers.google.com/analytics/devguides/collection/analyticsjs/cookies-user-id`, 2020.
[12] Custom dimensions & metrics. `https://support.google.com/analytics/answer/2709828?hl=en`, 2020.
[13] How to step through your code. `https://developers.google.com/web/tools/chrome-devtools/javascript/step-code`, 2020.
[14] Json.stringify(). `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify`, 2020.
[15] Tracking protection lists. `https://disconnect.me/trackerprotection`, 2020.
[16] Using http cookies. `https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies`, 2020.
[17] What are the values in _ga cookie? `https://stackoverflow.com/questions/16102436/what-are-the-values-in-ga-cookie`, 2020.
[18] Pieter Agten, Steven Van Acker, Yoran Brondsema, Phu H Phung, Lieven Desmet, and Frank Piessens. JSand: complete client-side sandboxing of third-party JavaScript without browser modifications. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 1–10. ACM, 2012.
[19] Quan Chen and Alexandros Kapravelos. Mystique: Uncovering information leakage from browser extensions. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1687–1700. ACM, 2018.
[20] Quan Chen, Peter Snyder, Ben Livshits, and Alexandros Kapravelos. Improving web content blocking with event-loop-turn granularity javascript signatures. *arXiv preprint arXiv:2005.11910*, 2020.
[21] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1388–1401, New York, NY, USA, 2016. ACM.
[22] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan Mayer, Arvind Narayanan, and Edward W Felten. Cookies that give you away: The surveillance implications of web tracking. In *Proceedings of the 24th International Conference on World Wide Web*, pages 289–299, 2015.
[23] Imane Fouad, Nataliia Bielova, Arnaud Legout, and Natasa Sarafijanovic-Djukic. Tracking the pixels: Detecting web trackers via analyzing invisible pixels, 2018.
[24] Gertjan Franken, Tom Van Goethem, and Wouter Joosen. Who left open the cookie jar? a comprehensive evaluation of third-party cookie policies. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 151–168, Baltimore, MD, 2018. USENIX Association.
[25] Balachander Krishnamurthy and Craig Wills. Privacy diffusion on the web: a longitudinal perspective. In *Proceedings of the 18th international conference on World wide web*, pages 541–550. ACM, 2009.
[26] Tobias Lauinger, Abdelberi Chaabane, Sajjad Arshad, William Robertson, Christo Wilson, and Engin Kirda. Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web. 2017.
[27] Sebastian Lekies, Ben Stock, and Martin Johns. 25 million flows later - large-scale detection of DOM-based XSS. In *20th ACM Conference on Computer and Communications Security Berlin 4.11.2013*, 2013.
[28] Sebastian Lekies, Ben Stock, Martin Wentzel, and Martin Johns. The unexpected dangers of dynamic javascript. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 723–735. USENIX Association, 2015.
[29] Jonathan R Mayer and John C Mitchell. Third-party web tracking: Policy and technology. In *IEEE Symposium on Security and Privacy*, pages 413–427. IEEE, 2012.
[30] Leo A Meyerovich and Benjamin Livshits. ConScript: Specifying and enforcing fine-grained security policies for JavaScript in the browser. In *IEEE Symposium on Security and Privacy*, pages 481–496. IEEE, 2010.
[31] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. You are what you include: large-scale evaluation of remote javascript inclusions. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 736–747. ACM, 2012.
[32] Xiang Pan, Yinzhi Cao, and Yan Chen. I do not know what you visited last summer: Protecting users from third-party web tracking with trackingfree browser. In *Proceedings of the 2015 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA*, 2015.
[33] Joe Gibbs Politz, Spiridon Aristides Eliopoulos, Arjun Guha, and Shriram Krishnamurthi. Adsafety: Type-based verification of javascript sandboxing. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 12–12, Berkeley, CA, USA, 2011. USENIX Association.
[34] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. Detecting and defending against third-party tracking on the web. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 12–12. USENIX Association, 2012.
[35] Iskander Sanchez-Rola, Matteo Dell'Amico, Davide Balzarotti, Pierre-Antoine Vervier, and Leyla Bilge. Journey to the center of the cookie ecosystem: Unraveling actors' roles and relationships. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2021.
[36] Philippe Skolka, Cristian-Alexandru Staicu, and Michael Pradel. Anything to hide? studying minified and obfuscated code in the web. In *The World Wide Web Conference*, pages 1735–1746, 2019.
[37] Steven Van Acker, Philippe De Ryck, Lieven Desmet, Frank Piessens, and Wouter Joosen. Webjail: least-privilege integration of third-party components in web mashups. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 307–316. ACM, 2011.
[38] Chuan Yue and Haining Wang. Characterizing insecure javascript practices on the web. In *Proceedings of the 18th international conference on World wide web*, pages 961–970. ACM, 2009.