Unsupervised Detection of Web Trackers

Unsupervised Detection of Web Trackers

Hassan Metwalley, Stefano Traverso, Marco Mellia
Politecnico di Torino, Italy
Email: firstname.lastname@polito.it

***Abstract*—When browsing, users are consistently tracked by parties whose business builds on the value of collected data. The privacy implications are serious. Consumers and corporates do worry about the information they unknowingly expose to the outside world, and they claim for mechanisms to curb this leakage. Existing countermeasures to web tracking either base on hostname blacklists whose origin is impossible to know and must be continuously updated. This paper presents a novel, unsupervised methodology that leverages application-level traffic logs to automatically detect services running some tracking activity, thus enabling the generation of curated blacklists. The methodology builds on an algorithm that pinpoints pieces of information containing user identifiers exposed in URL queries in HTTP(S) transactions.**

We validate our algorithm over an artificial dataset obtained by visiting the top 200 most popular websites in the Alexa rank. Results are excellent. Our algorithm identifies 34 new third-party trackers not present in available blacklists. By analyzing the output of our algorithm, some privacy-related interactions emerge. For instance, we observe scenarios clearly hinting to Cookie Matching practice, for which information about users' activity gets shared across several different third-parties.

I. Introduction

The last years witnessed the silent growth of web tracking services: collecting information about users' online activity is one of the most profitable activity in the Internet. There are hundreds of companies which base their whole business on it. A countless number of web tracking technologies are in use and tens of business models have been developed around web tracking [6], [14]. This phenomenon is ubiquitous, with both major and mostly unknown players taking part in it.

The web tracking practice raises many concerns about its implications on user's privacy. Indeed, its implementation results in leakage of information that users and companies would like to keep private: from sexual or religious preferences, to simple browsing histories. Many surveys have demonstrated that consumers and corporates want control over web tracking [9], [19]. Governments and policymakers have taken steps to intervene and advocated new technical approaches to enhance consumer choice about web tracking [7], [12]. Hence, there is a large ongoing effort to build technical countermeasures against web tracking. For instance, big players such as Mozilla have proposed their anti-tracking features [13]. Many plugins have been introduced to block interactions among the browser and tracking services. So far, the research community has focused on disclosing and quantifying the vastness of the problem [1], [3], [10], [15], [20], but only a few solutions have been proposed to curb this phenomenon [17], [18].

Simple actions such as blocking cookies are easily bypassed by web tracking services. For instance, a common workaround is to embed user identifiers in URL queries in HTTP requests. Alternatively, most of current web tracking countermeasures rely on blacklisting of tracking services and contents.However, how these blacklists are generated is unknown, and they are difficult to maintain over time. Finally, other solutions look too drastic to be embraced by the majority of users [17].

Similar in spirit to [4], this paper proposes a novel methodology to automatically generate curated blacklists that may be employed by any browser to block the web tracking services users encounter. Our methodology completely differs from [4] and complements it: while authors of [4] base their analysis on website DOM structures, our approach builds on the availability of application-level traffic logs, i.e., traffic traces reporting the information contained in the headers of HTTP transactions. This kind of logs may be automatically generated by browsing bots or crawlers, or shared by users in a crowd-sourced system. Based on the intuition that tracking services rely on per-user unique identifiers which browsers expose in the URL queries, we develop an unsupervised algorithm that analyzes URLs in HTTP request headers and seeks for pieces of information exhibiting a one-to-one mapping with the user profile generating the request. These pieces of information are identifiers contained in cookies, fingerprints, etc.

We validate our algorithm over a dataset of 868,000 HTTP logs we obtain by artificially browsing the top 200 most popular websites in the Alexa rank. Results are promising: our algorithm is effective at identifying a considerable amount of tracking services and the identifiers they employ to track users. For instance, we discover 34 never seen before tracking services. Interestingly, by analyzing how these identifiers are handled, some intriguing scenarios emerge. For instance, we observe cases of interactions among different tracking services suggesting the adoption of the Cookie Matching [2] practice; where the same user identifier is shared among different tracking services.

The proposed methodology is effective, yet simple, and can be employed by researchers, developers and practitioners to pinpoint tracking services in the web. Moreover, as it seeks for the user identifiers employed by web trackers, we believe our methodology is suitable for other contexts.

The remainder of the paper is structured as follows: Sec. II discusses the related work. Sec. III presents the proposed algorithm. Sec. IV describes the dataset and its collection procedure. Sec. V presents, first, the experiments we run to evaluate the parameter sensitivity of the algorithm, then, its capability at pinpointing user identifiers and web trackers together with some analysis examples. Finally, Sec. VI concludes the paper.

II. Related Work

Our study relates to measurement works about web tracking and online advertisement. Some of the notable works in this area are mostly oriented to understand which identifiers and techniques online tracking services exploit to record users' browsing activities. Authors of [20] examine the common identifiers trackers can leverage to identify users. Authors of [6] describe the techniques third-party trackers and popular online social networks employ to monitor the activity of their users. Other works provide an overview about how the web tracking phenomenon has worryingly grown in the last years [10], [11], [15]. Other works offer a globally distributed view on this phenomenon [5], [8].

As web tracking has raised many concerns about how it may affect users' privacy, many tracker-blocking applications, mostly being browser plugins, are available. They basically filter HTTP requests generated to tracking services. Ghostery[1] and DoNotTrackMe[2] are well know examples. These applications rely on blacklists built offline to prevent the browser to generate HTTP requests to web trackers. However, how such blacklists are produced is impossible to know.

Differently, the research literature about methodologies to counterfight the web-tracking practice is not rich. The most prominent works which specifically targets the problem are [18] and [4]. In the former, the methodology builds on a browser plugin called *ShareMeNot* that analyzes how trackers manage cookies through the browser. In a nutshell, this approach labels as trackers the owners of the pieces of code handling cookies and Adobe Flash plugins containing user identifiers. Differently, the authors of [4] explore the use of machine learning to analyze the structure of webpage code and identify web trackers.

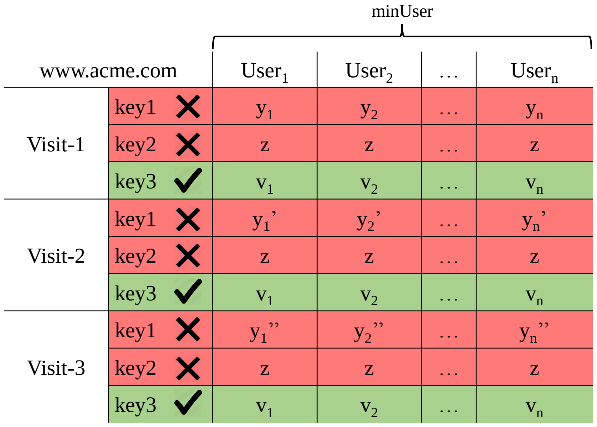
Our approach differs from these methodologies and, hence, can complement them. It simply builds on the observation that tracking services often bypass cookie-blocking policies by embedding user identifiers in URL queries contained in HTTP requests. Hence, its analysis is passive and only requires the availability of HTTP transaction logs. Second, it is unsupervised, as it does not require to know in advance the set of fields or keys containing user identifiers employed by tracking services. We are the first, to the best of our knowledge to explore this direction.

III. Automatic Detector of User Identifiers

In this section we present our automatic unsupervised methodology to pinpoint the user-tracking information exposed in HTTP request headers. We design an algorithm based on the observation that web services often exchange user identifiers as parameters in URL queries. We then look for possible parameters in HTTP GET requests that look like user identifiers.

Given a collection of logs **HS** aggregating the HTTP transactions generated by a known set of users and a targeted website domain W, our algorithm, illustrated in Alg. 1, extracts all HTTP key-value pairs contained in each HTTP request directed or referring to W, i.e., having W either in the `Host` (i.e., being first-party) or in the `Referer` field (i.e., being third-party). Consider for example `http://www.acme.com/query?key1=X&key2=Y`, where W is equal to `acme.com`, the algorithm extracts `key1` and `key2`, with values `X` and `Y`, respectively (lines 5-7 in Alg. 1). Then, for each key, the algorithm investigates biuniquenesses between the identifiers of the users generating the requests (e.g., the browser profile) and the values contained in the keys (lines 16-27). Intuitively, the algorithm looks for any key whose values are uniquely associated to the users, i.e., i) different for each different user, but ii) the same for the same user. Finally, to guarantee statistical evidence, the algorithm outputs the set of service-key pairs for which we observe at least $minUsers$ different user-value biuniquenesses (lines 28-33).

Fig. 1 reports an example of keys our algorithm processes for the target W=`www.acme.com`. Considering `key1`, it takes different values for different users, but these are not equal across visits, making `key1` a possible session identifier. `key2` maintains the same value across different users and visits. The key our algorithm elects as user-tracking is `key3`, as it is the only one whose values are different for different users, but do not change across different visits.



www.acme.com			minUser User$_1$	User$_2$	…	User$_n$
Visit-1	key1	✗	y_1	y_2	…	y_n
	key2	✗	z	z	…	z
	key3	✓	v_1	v_2	…	v_n
Visit-2	key1	✗	y_1'	y_2'	…	y_n'
	key2	✗	z	z	…	z
	key3	✓	v_1	v_2	…	v_n
Visit-3	key1	✗	y_1''	y_2''	…	y_n''
	key2	✗	z	z	…	z
	key3	✓	v_1	v_2	…	v_n

Fig. 1. Example reporting the kind of key our algorithm labels as user-identifier (`key3`), and examples of keys it discards (`key1` and `key2`).

Observe that, despite we focus on the user-tracking keys embedded in the URL queries of HTTP/S GET requests, our methodology can be easily extended to process the data the client transmits to the servers via POST requests, or embedded in the cookies. Similarly, for this study we focus on detecting single user-identifying keys, i.e., keys whose values alone show a one-to-one mapping with the user generating the requests. However, the algorithm can be improved to detect combinations of keys whose values may exhibit biuniqueness with the user.

IV. Dataset

To test our algorithm, we collect a dataset using a testbed based on Selenium WebDriver[3] to automatize the browsing of a selected set of websites. We browse the top 200 websites in the Alexa rank in the global category. Our browser installs a custom extension we build to log and dump to file all the HTTP and HTTPS request and response headers it observes. The scope of our analysis is pinpointing user-tracking keys, so

[1] https://www.ghostery.com

[2] https://www.abine.com/donottrackme.html

[3] http://www.seleniumhq.org

Algorithm 1 Automatic User Identifier Detector.

```
Input: HS, $W$, $minUsers$    #HTTP/S request log, the target website and the minimum number of distinct user-value pairs to observe.
Output: TS    #List of web services and their user-tracking keys for website $W$
$\mathbf{H_u}$ ← init_hash_table()    #Init hashtable of user identifiers
$\mathbf{H_v}$ ← init_hash_table()    #Init hashtable of key-value pairs
$\mathbf{H_k}$ ← init_hash_table()    #Init hashtable of host-key pairs
while h in HS do    #Read HTTP request logs
    $h$ ← uid, host, path, referer    #Extract fields of interest
    if $W$ in h.host or $W$ in h.referer then    #Check target is in the host or in the referer
        $K$, $V$ ← extract_keys(h.path)    #Extract keys and values from the path field
        while k, v in $K$, $V$ do    #Iterate all key names and values
            $host\_key\_uid$ ←create_hash(h.host,k,h.uid)    #Create hash for $\mathbf{H_u}$
            $host\_key\_value$ ←create_hash(h.host,k,v)    #Create hash for $\mathbf{H_v}$
            ADD_DISTINCT($\mathbf{H_u}$[$host\_key\_uid$],v)    #Insert all key-value pairs in $\mathbf{H_u}$
            ADD_DISTINCT($\mathbf{H_v}$[$host\_key\_value$],h.uid)    #Insert uid in $\mathbf{H_v}$
        end while
    end if
end while

while hash in $\mathbf{H_u}$ do    #Iterate over $\mathbf{H_u}$
    while value in $\mathbf{H_u}$[hash] do    #Iterate over values mapped to current hash
        if LEN($\mathbf{H_u}$[hash]) == 1 then    #Check current hash refers to one value only
            $host$, $key$, $uid$ ← decode_hash(hash)    #Decode hash into host, key and userID
            $hash_{aux}$ ← create_hash($host$, $key$, $value$)    #Create an auxiliary hash using host, key and value
            if LEN($\mathbf{H_v}$[$hash_{aux}$]) == 1 and $\mathbf{H_v}$[$hash_{aux}$] == $uid$ then #Check the auxiliary hash in $\mathbf{H_v}$ contains only one userID and check this corresponds to the one in $\mathbf{H_u}$
                $hash_{aux2}$ ← create_hash($host$, $key$)    #Create an auxiliary hash using host and key
                INCREMENT($\mathbf{H_k}$[$hash_{aux2}$])    #Increment the number of unique userID-value pairs for this key
            end if
        end if
    end while
end while

while hash in $\mathbf{H_k}$ do    #Iterate over $\mathbf{H_k}$
    if LEN($\mathbf{H_k}$[$hash$]) $>=$ $minUsers$ then    #Check current hash contains at least $minUsers$ different userID-value pairs
        $host$, $key$, ← decode_hash(hash)    #Decode hash into host and key
        ADD(TS, $host$, $key$)    #Add host and key to the output list
    end if
end while
```

to avoid misclassifying keys which may identify the session (i.e., the visit), we need each website to be visited multiple times by the same user. We instrument the browser to create a new user profile, visit the 200 websites in the list, and log all HTTP/S transactions it generates. We repeat this procedure 14 times, so to have 14 traces related to 14 different users. Thus, for each user profile we perform three rounds of visits at different times. In total, we perform 9,000 visits allowing us to collect the logs of more than 868,000 HTTP/S transactions.

In order to let anyone reproduce the experiments presented in this paper, we make our dataset available to the community[4].

V. Results

In this section we first run experiments to investigate the impact of parameters choice may have on the results. Then, we provide proofs about the algorithm effectiveness and examples of analysis we can run over its output.

A. Parameter Sensitivity

We recall $minUsers$ is the minimum number of unique user-value pairs the algorithm must observe to label a key as user identifier. In particular, we check how the number of returned keys which our algorithm classifies varies when increasing $minUsers$. Naively, one may think to set $minUsers$ to be large, because, if too low, we expect to misclassify those keys that may instead contain other kind of information, such as, e.g., session identifiers. In other words, a small $minUsers$ may increase the number of false positives. On the other hand, a too large $minUsers$ could cut out legit positives associated to portals which embed a large set of third-party objects that may not be always present. For instance, some users may access a news portal at the moment it embeds a third-party advertisement ad_i using a given user-identifying key, k_i, but other users accessing the same portal may encounter a different advertisement service, ad_j, and, thus, a different key k_j. In this case the population of users gets split in two halves, and a too large $minUsers$ would filter both of them out from the set of true positives.

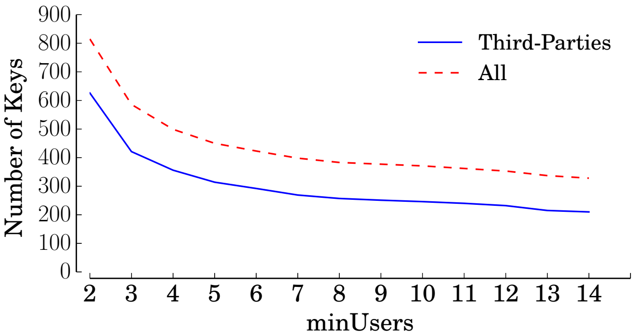


Fig. 2. Number of user-tracking keys pinpointed by the algorithm when increasing parameter $minUsers$ and considering HTTP requests to third-parties only (red dotted curve) and to both first- and third-parties (blue solid curve).

We run an experiment to evaluate the trade-off value for $minUsers$ which guarantees a reasonable accuracy while not

[4] It can be downloaded at https://www.dropbox.com/s/elytxdz9h9ue3q4/MegaTrace.gz?dl=0.

Web service	Keys
www.walmart.com	`customerId`
omniture.walmart.com	`vidn,c17`
beacon.walmart.com	`btc,ezakus_id,dwtc,visitor_id`
dis.criteo.com	`google_gid,CriteoUserId`
cm.g.doubleclick.net	`CriteoUserId,cb`
dw.wmt.co	`btc,dwtc`

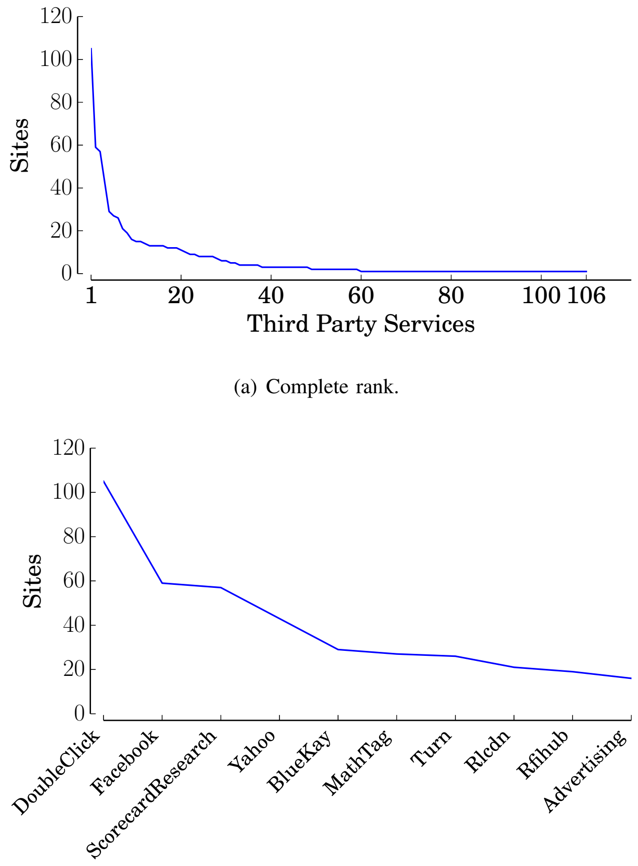
TABLE I. THE USER-TRACKING WEB SERVICES AND THE KEYS OUR ALGORITHM DETECTS FOR WEBSITE *www.walmart.com*.

cutting out legit true positives. Fig. 2 reports the number of user-identifying keys our algorithm identifies when we set different $minUsers$ values to process all the requests $\mathbf{HS}$ in our dataset. We consider both the cases in which the algorithm processes the set of HTTP requests to third-party services only – services embedded in websites whose HTTP requests show a mismatch between the hostnames contained in `Host` and `Referer` fields – (red dotted curve), and all the requests (i.e., taking into account both first- and third-parties) in the dataset (blue solid curve). As expected, the number of keys increases when $minUsers$ is small: we count more than 600 tracking keys used by third-party services when we rely on just two distinct user-value pairs to label a key as user-identifying. Observe that the number of keys keeps decreasing when $minUsers$ increases. For third-parties the number of keys labelled as user-tracking decreases to 210 when $minUsers$ equals 14, and to 328 when considering both first- and third-parties. We manually verify the data and we observe that the pool of third-party web services associated to the same website actually changes between different visits, thus confirming the intuition described in the previous paragraph. Hence, as a counterproof, we run a second experiment: first, we make sure of focusing on a set of services for which we observe some visits for each of the 14 users we have in our dataset. 14 is the maximum value we can use, as this is the number of distinct browsing profiles we employ in our collection. Given the resulting subset of services, we filter $\mathbf{HS}$ to keep only the requests pointing to them, thus obtaining a smaller dataset, $\mathbf{HS}_{Users=14}$. Then, we use this latter to run again the algorithm varying $minUsers$. This time we observe that the number of keys stabilizes at 328 when $minUsers \geq 6$, while some false positives (keys associated to services in $\mathbf{HS}_{Users=14}$, but carrying session identifiers mostly) are found for values of $minUsers < 6$. Impact is minimal, but present. Setting $minUsers = 6$, we are confident of correctly labelling a key as user-identifying, while on the other hand we do not filter out too dynamic web services actually implementing some user-tracking feature.

B. Identifying User-Tracking Services

We now analyze the results we obtain by running the algorithm over our artificial dataset.

First, we provide an example of the output of the algorithm. We run the algorithm on the portion of the dataset referring to the website *www.walmart.com*, i.e., when considering all HTTP transactions whose `Host` or `Referer` contain *walmart.com*. Table I reports the resulting hostname and key names discovered that first- (top) and third-party (bottom) services associated to *walmart.com* use. For this experiment we set the algorithm parameter $minUsers$ to equal 6, as chosen in the previous section. Unfortunately, we lack a proper ground-truth to evaluate the accuracy of our algorithm. However, the results we obtain are reasonable. As shown, the algorithm identifies 3 distinct third-party trackers linked to *walmart.com*. They are well-known trackers found in blacklists of ad-blocking services. Furthermore, key names clearly suggest the exchange of possible user-tracking identifiers. Notice also that some trackers embed keys containing the name of other trackers, e.g., *cm.g.doubleclick.net* use the key `CriteoUserId`. As we will describe in the Sec.V-D, this is the result of the Cookie Matching practice [2].



(a) Complete rank.

(b) Top-10 rank.

Fig. 3. The rank of third-party trackers detected by our algorithm sorted by the number of sites (first-parties) they appear associated to.

C. Third-Party Tracker Analysis

The result presented in Fig. 2 shows that both first- and third-parties do employ keys to track users. Indeed, when $minUsers$ equals 6 we observe that more than 130 keys are employed by 121 different first-party services, and more than 300 user-identifying keys are associated to third-party services. Intrigued by such a wide pervasiveness, in this section we focus on the detection of services in the latter category. Hence, we run the algorithm over our whole artificial dataset. We obtain a list containing more than 100 third-party services using some user-identifying key. Fig. 3(a) reports the rank of third-party trackers sorted by the number of first-party services they are associated to, i.e., their popularity across different first-party services. Fig. 3(b) details the top-10 most popular third-party trackers. As shown, top 10 third-party trackers appear to be associated to 20 or more first-parties (out of the 200 we take from the Alexa rank), and most of the third-party trackers cover a very limited number of first-party services. More than 40 trackers cover one service only. The most pervasive third-party is DoubleClick which is present in 52% of the 200 considered services.

We compare our output with the tracker list we have built by merging together data we obtain from different sources as described in [15]. This list contains a total number of 443 different tracking services and can be considered as a verified ground-truth. The number of trackers identified by our algorithm is smaller (106) because of the very limited amount of websites we consider to build our dataset, 72 of them are present in the list, and are thus verified, 34 of them are new and do not appear in the list. Manually checking, we validate that they are all actual third-party trackers. This result, even if not conclusive, is extremely encouraging, and lets us be confident about the novelty of our perspective and the effectiveness of our approach.

D. Interesting Findings

In the following we present some interesting findings that emerge when analyzing the user-identifying keys returned by our algorithm and the values they contain. More in detail, we observe many cases in which the same value, i.e., the unique piece of information associated to a user, is contained in user-identifying keys used by *different* services.

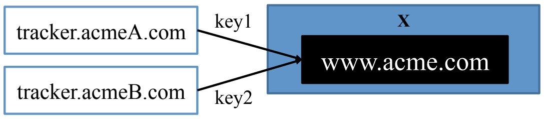
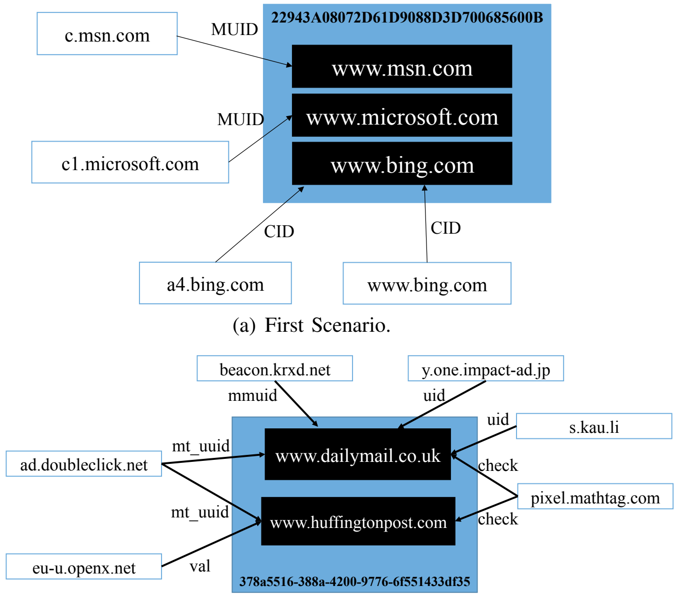


Fig. 4. Schema example.

To represent these interactions, we employ the schema in Fig. 4: `www.acme.com` is the visited website. `tracker.acmeA.com` and `tracker.acmeB.com` are both services labelled as trackers by our algorithm. `key1` and `key2` are the tracking keys they respectively employ to identify users. `X` is a user identifier key value (can be, e.g., a hash contained in a cookie) we pick from our dataset and contained in both `key1` and `key2`. Surprisingly, both `key1=x` and `key2=x`, despite `key1` and `key2` are "independently" generated by `acmeA` and `acmeB`. Clearly this pinpoints to some collision between the two.

We observe three main scenarios in which user identifiers are shared across several services.

The simplest scenario is similar to the example depicted in Fig. 5(a). In this case, a user accessing the first-party services *www.bing.com*, *www.msn.com* and *www.microsoft.com*, administrated by the same corporate (Microsoft), is tracked by the services *c1.microsoft.com*, *a4.bing.com*, and *c.msn.com* (still administrated by Microsoft) which use different keys, `MUID`, `CID` and `MUID`, respectively, to exchange the same user identifier value. Being the user identifier shared among services under the same corporate umbrella, this suggests a tracking platform administrated by the same organization, i.e., Microsoft in our case. This case does not appear controversial from a privacy perspective.



(a) First Scenario.

(b) Third Scenario: Cookie Matching and Real-Time Bidding.

Fig. 5. Scenarios in which the same user identifier is shared among several services.

The second interaction example is very similar to schema example in Fig. 4 and is not figured for brevity. In this case, a user accessing the first-party service *www.walmart.com* is assigned an identifier employed by the third-party services *cm.g.doubleclick.net* and *dls.criteo.com*, and contained in the key `CriteoUserId`. Notice two substantial differences with respect to the scenario depicted in Fig. 5(a): first, the same user identifier is shared among two different third-party services not belonging to the same owner. Second, notice that DoubleClick employs a key clearly provided by Criteo, a well known tracking company. This kind of interaction is the typical result of a practice, introduced by Google and named Cookie Matching [2], that allows two separate parties to synchronize their users' identifiers. For example, typically, a user is assigned cookies from the several parties she encounters during her browsing activity. Hence two trackers normally assign their own distinct cookies to the same user. Thanks to the Cookie Matching mechanism, one or both of them will have these cookies mapped to each other. Cookie Matching constitutes a fundamental part of the Real-Time Bidding (RTB) mechanism [16]. RTB is a common web advertising technique which implements real-time automatic auctions. Typically, a website enabling RTB, called *seller* in RTB terminology, aims at selling the advertisement spaces available on its page for the best offer. To enable the auction, two other kinds of third-parties are involved: the *auctioneer*, that orchestrate the auction, and the *buyers*, which generate bids for the advertisement spaces. When a user visits the seller website, the auctioneer service collects the identifiers contained in cookies from different buyers and run the Cookie Matching protocol. Once the user identifier is synchronized among the auction participants, the auctioneer collects the buyers' bids and elects the winning buyer. Hence, this latter will be authorized to provide the content to fill the advertisement space.

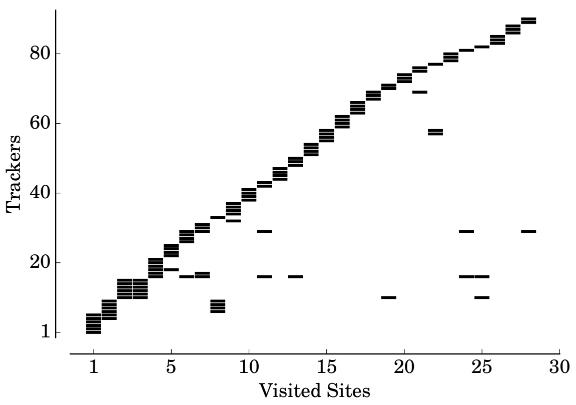


Fig. 6. Trackers (y axis) involved in Cookie Matching actions and embedded by the websites (x axis) we visited to build our dataset.

The last example of interaction we observe in our dataset is depicted in Fig. 5(b). This scenario hints to a practice which combines Cookie Matching and RTB. We observe that the same user identifier is shared between two sellers, *www.dailymail.co.uk* and *www.huffingtonpost.com* (which are governed by the same owner), the auctioneer, DoubleClick, and five different buyers. Although RTB and Cookie Matching are acclaimed by the advertising industry, their implementation leads to scenarios in which user identifiers are handled by different players not governed by a common authority. We believe this cross-parties access to users' data looks boggling and raises considerable worries about their implications on users' privacy [16].

We conclude our analysis by providing a more general overview about the cases in which user identifiers are shared among multiple parties, i.e., in which the practice of Cookie Matching takes place. We consider all the websites in the dataset. We then keep those for which we observe some Cookie Matching practice involving at least 3 distinct third-parties. We report the result in Fig. 6. It shows the websites sorted by the number of trackers they embed and involved in some Cookie Matching action (larger numbers to the left). Each black block represents the occurrence of a user identifier being shared. First, we observe that out of the 200 visited websites, 30 of them host at least 3 services involved in Cookie Matching. The number goes up to 76 if we decrease the number of involved parties to 2. Second, we notice five cases in which Cookie Matching may involve up to 6 different trackers (see the first five columns on the left), i.e., where 6 different third-party entities share the same view on users' activity. This is another striking finding which calls for further investigation about how user data is spread in the web tracking ecosystem.

VI. Conclusion

Motivated by the attention that online tracking services have recently attracted in the scientific community because of the implications they have on users' privacy, we developed a novel, unsupervised methodology based on an algorithm which inspects URL queries in HTTP(S) requests and seeks for the pieces of information exhibiting a one-to-one mapping with the user generating the requests. The algorithm outputs a list of first- and third-party web services which employ any user-tracking keys.

We evaluated the algorithm sensitivity of its only required parameter and validated its output by running it against a dataset of HTTP(S) logs we obtain by visiting three times the top 200 most popular websites in the Alexa rank, and using 14 different user profiles.

We showed that the algorithm is effective at automatically scouting tracking services. Even if considering our limited dataset, it could detect 34 never seen before third-party trackers. Finally, we showed examples of findings that emerge when analyzing the output of the algorithm. We pinpointed interaction scenarios which hints to Cookie Matching and Real-Time Bidding.

The methodology presented in this paper complements existing techniques for tracker detection and it can help researchers and developers to build and maintain more curated tracker block lists.

References

[1] Data Transparency Lab, http://datatransparencylab.org/.

[2] Google. Googles cookie matching protocol. https://developers.google.com/ad-exchange/rtb/cookie-guide.

[3] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *ACM SIGSAC*, 2014.

[4] J. Bau, J. Mayer, H. Paskov, and J. C. Mitchell. A promising direction for web tracking countermeasures. In *IEEE W2SP*, 2013.

[5] C. Castelluccia, S. Grumbach, and L. Olejnik. Data Harvesting 2.0: from the Visible to the Invisible Web. In *WEIS*, 2013.

[6] A. Chaabane, M. A. Kaafar, and R. Boreli. Big Friend is Watching You: Analyzing Online Social Networks Tracking Capabilities. In *ACM WOSN*, 2012.

[7] F. T. Commission et al. Protecting consumer privacy in an era of rapid change. *FTC Report, Washington, DC*, 2012.

[8] M. Falahrastegar, H. Haddadi, S. Uhlig, and R. Mortier. The Rise of Panopticons: Examining Region-Specific Third-Party Web Tracking. In *TMA*. 2014.

[9] C. Hoofnagle, J. Urban, and S. Li. Privacy and modern advertising. In *Amsterdam Privacy Conference*, 2012.

[10] B. Krishnamurthy, K. Naryshkin, and C. E. Wills. Privacy leakage vs. Protection measures: the growing disconnect. In *IEEE W2SP*, 2011.

[11] B. Krishnamurthy and C. Wills. Privacy Diffusion on the Web: A Longitudinal Perspective. In *WWW*, 2009.

[12] N. Kroes. Online privacy-reinforcing trust and confidence. *speech, Brussels, June*, 22, 2011.

[13] J. Mayer. The new firefox cookie policy. *XRDS*, 20(1):16–17, Sept. 2013.

[14] J. Mayer and J. Mitchell. Third-party web tracking: Policy and technology. In *IEEE SP*, pages 413–427, May 2012.

[15] H. Metwalley, S. Traverso, M. Mellia, S. Miskovic, and M. Baldi. The online tracking horde: a view from passive measurements. In *TMA*, 2015.

[16] L. Olejnik, M.-D. Tran, and C. Castelluccia. Selling off Privacy at Auction. In *ISOC NDSS*, 2014.

[17] X. Pan, Y. Cao, and Y. Chen. I Do Not Know What You Visited Last Summer: Protecting Users from Third-party Web Tracking with TrackingFree Browser. In *ISOC NDSS*, 2015.

[18] F. Roesner, T. Kohno, and D. Wetherall. Detecting and Defending Against Third-party Tracking on the Web. In *USENIX NSDI*, 2012.

[19] J. Turow, J. King, C. J. Hoofnagle, A. Bleakley, and M. Hennessy. Americans reject tailored advertising and three activities that enable it. *Available at SSRN 1478214*, 2009.

[20] T.-F. Yen, Y. Xie, F. Yu, R. P. Yu, and M. Abadi. Host Fingerprinting and Tracking on the Web: Privacy and Security Implications. In *ISOC NDSS*, 2012.