Tracking the Trackers

Zhonghao Yu
Cliqz
Arabellastraße 23
Munich, Germany
zhonghao@cliqz.com

Sam Macbeth
Cliqz
Arabellastraße 23
Munich, Germany
sam@cliqz.com

Konark Modi
Cliqz
Arabellastraße 23
Munich, Germany
konarkm@cliqz.com

Josep M. Pujol
Cliqz
Arabellastraße 23
Munich, Germany
josep@cliqz.com

ABSTRACT

Online tracking poses a serious privacy challenge that has drawn significant attention in both academia and industry. Existing approaches for preventing user tracking, based on curated blocklists, suffer from limited coverage and coarse-grained resolution for classification, rely on exceptions that impact sites' functionality and appearance, and require significant manual maintenance. In this paper we propose a novel approach, based on the concepts leveraged from k-Anonymity, in which users collectively identify *unsafe* data elements, which have the potential to identify uniquely an individual user, and remove them from requests. We deployed our system to 200,000 German users running the Cliqz Browser or the Cliqz Firefox extension to evaluate its efficiency and feasibility. Results indicate that our approach achieves better privacy protection than blocklists, as provided by *Disconnect*, while keeping the site breakage to a minimum, even lower than the community-optimized *AdBlock Plus*. We also provide evidence of the prevalence and reach of trackers to over 21 million pages of 350,000 unique sites, the largest scale empirical evaluation to date. 95% of the pages visited contain 3rd party requests to *potential* trackers and 78% attempt to transfer *unsafe* data. Tracker organizations are also ranked, showing that a single organization can reach up to 42% of all page visits in Germany.

Keywords

Tracking, Data privacy, k-Anonymity, Privacy protection

1. INTRODUCTION

Online privacy is a topic that has already become mainstream. General media talks about it [21, 16, 12], governments have passed legislation [10], and perhaps more revealing is that people has started to take active steps towards protecting their privacy online. According to [30] ad-blocking usage grew by 70% in 2014, culminating in 41% of people aged between 18-29 using an ad-blocker. This figure is consistent with the results of the empirical evaluation of 200,000 users in Germany presented in this paper.

Any person browsing the Web today is under constant monitoring from entities who track the navigation patterns of users. Previous work [19] reported that 99% of the top 200 news sites, as ranked by Alexa, contain at least one tracker, and at least 50% of them contain at least 11. The presence of trackers in popular news domains might be expected—these domains are monetized by advertisement—however that does not make this loss of privacy any less dramatic.

The loss of privacy is not restricted to a particular area the Web, such as news sites. The results of our large scale field study provide evidence that 84% of sites contained at least one tracker sending *unsafe* data. Out of 350K different sites visited by 200K users over a 7 day period, 273K sites contained trackers that were sending information that we deemed *unsafe*. Data elements that are only and always sent by a single user, or a reduced set of users, are considered *unsafe* with regard to privacy.

Companies running trackers have the ability to collect individual users' browsing habits, not only on popular sites, but virtually across the whole Web.

Tracking however has very legitimate use cases: Advertisers need to know how effective targeting is and profiles are needed to serve tailored ads. Social buttons and other widgets are convenient ways to share content. Social logins help to consolidate an online identity and reduce the overhead of managing passwords. Analytics tools like *Optimizely* or *Google Analytics* help site owners to better understand the traffic they receive, while other analytics like *New Relic* give site owners valuable insight on performance.

The Web, as of today, has evolved to become an ecosystem in which users, site-owners, network providers and trackers coexist. One cannot consider trackers as mosquitoes that can be wiped out without consequences; US revenues from online advertising in 2013 reached $43 billion [17], and trackers play an important role in this figure. If all 3rd party traffic were to disappear or be blocked overnight, there would be significant disruption to the ecosystem. On the other hand, revenues cannot be used as an excuse if it comes at

WWW 2016, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4143-1/16/04.
http://dx.doi.org/10.1145/2872427.2883028.

the expenses of the loss of privacy of the users. Privacy is a right that should be protected [37, 35].

In this paper we propose a step further towards the solution of this problem. Unlike other anti-tracking systems like *Disconnect*, *Privacy Badger*, *Ghostery*, *Adblock Plus*,*NoScript* and *Firefox Tracking protection* which are already quite popular, the system described in this paper does not rely on blanket blocking of 3rd parties based on blacklists. Instead, our method, inspired by k-Anonymity [38] with some tweaks to limit known caveats, filters only the data elements sent to trackers that are not considered *safe*, where safety is determined by consensus among all users running our anti-tracking system.

The advantages of the method presented in this paper are two-fold: 1) it is friendlier to trackers, data needed to run their services will continue to exist, unless of course, it is considered *unsafe*. And, 2) it offers better coverage, less false positives and faster detection than systems based on curated blacklists. Inspecting the trackers data collaboratively in real-time results in better protection for the users as we will see in detail in Section 5.

An additional contribution of this paper is empirical evaluation of the trackers' prevalence and reach in the wild. 200,000 Cliqz users have been tracking the behavior of all trackers present in all sites they visit, which constitutes the largest field study about tracking to date, summarized in Section 3.

2. MOTIVATION

This section will cover basic concepts of tracking as well as an overview of the state of the art. For clarity, we will use real-world examples, but without losing of generality.

2.1 Dissecting Tracking

Tracking is a mechanism to record certain browsing patterns of a user in order gain information. A priori there is nothing wrong with it. If a user visits a travel site and then starts to receive advertising about hotels on her favorite news site, one could argue that this is actually a use-case that benefits the user. The re-targeting described does not imply a loss of privacy by default. The loss of privacy comes as a result of how tracking is typically implemented.

A example of a typical implementation is the travel site *kayak.de* and the news site *huffingtonpost.co.uk*, which share a tracker owned by *Bluekai*. There is a piece of javascript code (`tags.bkrtx.com/js/bk-coretag.js`) that gets executed on the user's browser every time they visit any page from either *kayak.de* or *huffingtonpost.co.uk*.

This piece of javascript typically sends the source page s (the page being visited) in the *HTTP-referer* as well as the following information:

```
bklc=55f6ad4d
l=https://www.kayak.de/
ua=f82610bef1d54776cde605b90b0c7949
t=1444203542439
m=020810a3483fc8307caa483fd192bc02
lang=07ef608d8a7e9677f0b83775f0b83775
sr=1440x900x24
cpu=4b4e4ecaab1f1c93ab1f1c93ab1f1c93
platform=6d44fad93929d59b3929d59b3929d59b
plugins=d4de4a68c91685d0ff4838ce3714359a
cn=df62ddfcfa96f717f2ee5a7d912e7102
```

Among this data we can see the string `55f6ad4d` which most likely identifies uniquely the user, acting as a *uid*. We say *most likely* because there is no way to know with absolute certainty. **What we do know however, is that this particular value has only been seen by this user out of a very large population and it does not change when the users visits different sites, therefore, whether it is a user unique identifier (*uid*) or not is somewhat irrelevant. If it was not, or was not intended to be, it could still be used as one.** Detecting such values is the core of the anti-tracking system presented in this paper.

From the data, the tracker *Bluekai* has the ability to learn the relationship (u, s), meaning user u visited page s. Besides *kayak.de* and *huffingtonpost.co.uk*, *Bluekai* is on almost 4,000 other sites. That means that *Bluekai* could learn a sizable chunk of the browsing history of a given user, and likely without their knowledge.

The proportion of the a user's history a tracker can eventually learn depends on the *reach* of the tracker, i.e. the percentage of site-owners using the tracker's javascript on their sites. If a tracker has only a small *reach*, then there is little risk. However, our analysis of tracker reach (see Section 3) shows that this is rarely the case. *Bluekai*, for example, can track users across 1.3% of all sites and 3.4% of the pages visited by 200,000 users. Trackers can learn a significant portion of the user's browsing, e.g. do-it-yourself portals, porn sites or medical forums.

At this point two important questions arise: 1) why do site-owners agree to put such code on their sites? And 2) do trackers really want to know all the information about a user? Unfortunately we cannot provide an answer accounting for all the dimensions, but we can share our expertise and learnings from interacting with trackers.

2.1.1 Why site owners use trackers?

The Web has evolved to become a Software as a Service mash-up, where site-owners tend to outsource certain functionalities to 3rd parties. These service providers offer analytics of visits, site performance, social widgets, content aggregation, comments systems, content delivery, and targeted advertising, just to mention some. The side-effect of such services is that 3rd parties have the ability to run their code in the user's browsers, and consequently, gain the ability to track them.

Take for instance *Facebook*'s share button. Site owners hope that ease of sharing will translate to more traffic to their sites. If this is not reason enough, there is often also the users' expectation for sites to have a share button. For one reason or another site owner might eventually add the share button. When that happens *Facebook* has the ability to become a tracker, as protrayed in the following example.

Tracking from a free widget: every time you visit a page with a *Facebook* component, they will receive the page URL via *HTTP-referrer* as well as the following extra data,

```
datr=_zr8VGU5cOvsTE_CjXTxF9
lu=TTA08XEc9ieLocEDius7A
fr=0SoRz_o5WZz6ioQ.BV5h.WE.FYS.0.AWZSMd
c_user=100002835278978
```

via Cookie when you are logged in. In this case your *uid* is explicit and known: it is the `c_user` parameter. When you logout, the `c_user` parameter is removed but the other

data remains[1]. It is difficult to assess whether the remaining data is privacy sensitive or not. However, by comparing data across multiple users, we find that

```
datr=_zr8VGU5cOvsTE_CjXTxF9
```

is actually seen once and only once per user in a very large population. Therefore, `datr` might not be an intended *uid*, but it can be used as such[2].

We cannot determine what is the intended use of `datr` but this question, as already mentioned before, is irrelevant. The data element is *unsafe*, regardless of its function, because it can be associated to a single user, and consequently, should not be sent to *Facebook*. This uniqueness may be unintentional—an unexpected side-effect of some other functionality. In such cases we could argue that privacy protection systems like ours do not only benefit users but also benefit trackers since it offers a guarantee that they are not *unintentionally* collecting user identifiers.

2.1.2 Why trackers track the way they do?

So why do *Facebook* and *Bluekai* send a data element that can be used as a *uid*? Sometimes the motivation may be that they want to learn the user's browsing behavior, but often it is just a by-product of a particular technical or implementation choice.

In the case of share buttons, displaying personalized information in the context of an external page requires a *uid* to identify whose information needs to be retrieved. Likewise, *Bluekai* requires knowledge of users' intent or interests in order to re-target advertising effectively. To do this *Bluekai* builds a profile for the user' browsing history, using a *uid* as a foreign key to group data by user.

The fact that the need for a *uid* can be justified due to technical choices does not make it less problematic with regard to privacy. As a matter of fact, we hope that the pressure of anti-tracking systems, including the one presented in this paper, will motivate trackers to rethink their service's design so that users' privacy is not compromised as an unfortunate by-product of a particular implementation, especially given viable alternative approaches [39, 15, 34].

We stress that in all of the examples in this paper, we state that companies have the *ability to* track, but this does not imply guilt on their part. Although trackers in general have the ability to learn private associations, like a user u visiting a site s, this does not mean that they will keep this data forever, or that they will misuse it in any way. Data might be deleted upon receipt, or only used in real-time and never stored. This, however, remains a matter of trust.

We believe the vast majority of companies collecting such data are to be trusted. However as long as *unsafe* data is received there is always a possibility of a privacy breach. Besides hacking, vulnerabilities include disgruntled employees [6], companies going bankrupt and selling their assets [32] or plain and simple government intervention [14, 26].

2.2 Related work on Tracking Protection

Much effort has been put into tracking protection. There are many different systems which can be categorized in several groups depending on their *modus operandi*. However they all aim for the same goal: to prevent 3rd party trackers from learning information about the user. Note that some systems provide anti-tracking as an additional feature because is related to its main goal. *AdBlock Plus*'s primary focus, for instance, is blocking advertisement.

Noscript [29] takes the most conservative approach with regard to privacy: it prevents any javascript code from being executed in the browser. This blocks most forms of communication between the user's browser and the 3rd party, however, this approach can cause a lot of problems with user experience since modern sites rely heavily on javascript.

Privacy needs to be preserved but there is an obvious trade-off between privacy and usability. As a matter of fact this is one of the main issues of anti-tracking, to protect privacy while not breaking, or, if that is not possible, minimizing the breakage of the site's appearance and functionality. This trade-off has some interesting ramifications.

Minimizing site breakage is not only needed for user satisfaction and retention, but it also has an indirect impact on the privacy protection. If sites break, users might attempt to solve it by adding exceptions and by creating custom rules to the anti-tracking engine. These rules may open gaps in their security protection, at least for the majority of users who may not really understand the underlying implications of the whitelisting or the exceptions they are adding.

We would like to add that besides minimizing malfunctions on the user side, these systems should also try to minimize malfunctions on the tracker side too.

The most common approach to anti-tracking is the use of curated blocklist of trackers, effectively blacklists. *Ghostery* [13], *Disconnect* [7] *Adblock Plus* [31], and beta *Firefox Tracking protection* [25] use this approach, where communication between the user's browser an a 3rd party in the blocklist is blocked (limited to Cookies or to the whole HTTP request).

Creating and maintaining precise and up-to-date blocklists is a difficult proposition, as past experience in areas like Spam and Phishing detection systems reported [36, 20]. In order to classify 3rd parties as trackers, one firstly needs to be aware of new domains that appear, and secondly one must regularly re-evaluate that known domains have not changed their behavior. There are however at least two further problems with the blocklist approach:

1) Blocklists typically operate at the level of domain suffix[3]. Once the 3rd party is classified as tracker it will get blocked. This approach however is very coarse-grained. Our empirical evidence shows (see Sect. 5.1) that about 78% of requests to 3rd parties blocked by *Disconnect*'s blocklist exhibit a mixed behavior, meaning that sometimes the content should be blocked and sometimes not. Domain name does not offer the proper resolution for an accurate classification. While the 78% figure is novel, the problem was known. In fact it is the underlying reason why blocklist based systems are evolving to offer a finer resolution control by extending blocklists to accept regular expressions and other case based exceptions. Why would finer grained resolution matter? It would be reasonable to assume that if a 3rd party is tracking for some cases it should be blocked even when if not. This assessment is however incorrect because of the aforementioned trade-off between site breakage and protection; too aggressive blocking produces site-breakage which

[1]Study [3] commissioned by the Belgian Privacy Commission found that *Facebook* is tracking users when opted out.
[2]Perhaps the parameter used to hack into accounts [4].

[3]although there are some which increase the resolution to full URL or some sophisticated regular expressions

if not handled, will force users to abandon anti-tracking or to create rules that subvert protections.

2) The other problem with blocklists is whether one can determine if the behavior of a 3rd party warrants inclusion in the blocklist. The fact that trackers have a mixed tracking behavior makes this evaluation even more difficult, given that the case of pure behavior is already a difficult task. Catching a tracker involves analyzing dozens or hundreds of requests trying to find a pattern or data element that is considered suspicious. Furthermore, this behavior represents an ever-moving target, for example the crackdown on 3rd party cookies has led to the rise of a multitude of other tracking methods to replace them, e.g. local storage, etags, fingerprinting, HTML5 canvas [8, 23, 1]. Every time a new method to fingerprint is detected, 3rd party trackers that were considered safe should to be reanalyzed for safety. These issues put those working with blocklists to protect users' privacy at a serious disadvantage in the *arms-race* against trackers. Additionally, many times data elements that could be *uid*s will simply escape detection, as we will see in Sect. 5.3.

Due to the coarse-grain resolution of blocklists, and the difficulty of detecting a 3rd party by requests inspection, we decided to explore a more algorithmic solution. In this respect ours is not the first attempt: *Privacy Badger* [9] relies on heuristics to identify tracking domains, however, it seems that in practice their approach alone leads to a significant amount of site breakage [11, 18, 19]. We rely on global information, where users cooperate on tracking the trackers. Another important difference in our approach is that we do not focus on identifying and blocking trackers but rather on deciding which pieces of data sent to 3rd parties are *safe* or *unsafe* with regard to privacy. The empirical evaluation of our method in Sect. 5 shows greater protection coverage than state-of-the-art methods based on blocklists, while keeping site breakage at bay.

3. TRACKING IN THE WILD

Before describing our system let us present our findings on the prevalence and reach of trackers on the Web.

Using the anonymous data needed to collectively evaluate *safeness* (see Sect. 4.1) we can get a comprehensive picture of the prevalence and reach of trackers for 350,000 different sites, visited by different 200,000 German users for a full week period[4]. This is the largest empirical evaluation to the best of our knowledge. Previous work focused on studying tracking for a subset of sites, e.g. the top 200 sites according to Alexa or on a small population[5] and provided evidence that trackers are present in 99% of the most popular News sites. Acar et al. [1] analyzed a much larger set of 100K sites to characterize tracking methods, the study however did not include real users (traffic was automated using *Selenium*). Our results reproduce previous findings as well as provde and an answer to the question of whether tracking is localized on specialized domains, like popular and heavy monetized sites, or it is a wide-spread phenomena that affects the whole Web.

3.1 Prevalence of Trackers

For seven days we observed 200,000 different users visiting roughly 21 million Web pages. The visits went to 5 million unique pages (URLs) spanning over 350,000 unique sites (domains). Figure 1a shows the number of 3rd parties that are classified by our method as *potential* trackers for each page load (a user visiting a page/URL) as well as for each unique first party site (domain). Figure 1b shows the same relationship but counting only the cases when we have seen evidence of an attempt to send *unsafe* data to one of those *potential* trackers.

In Figure 1a we can see that 95% of page loads produce a request to a *potential* tracker, i.e. a 3rd party that exhibits a connectivity pattern typical for a tracker[6]. 24% of the page loads result in requests to at least 10 of these entities.

As we already mentioned in the discussion about tracker blocklists in Sect. 2.2, trackers often have a mixed behavior, and not all 3rd party requests to trackers contain privacy sensitive data. Therefore, we would like to provide a more precise picture of the amount of page loads subjected to tracking. In Figure 1b we only consider page loads that produce a request to a *potential* tracker and that the request contains data considered *unsafe*, i.e. data elements that are user identifiers or can be used as such (see 4.1 for a formal definition). With this much stricter constraint, the number of page loads with no tracking involved increases to 22%, which means that the remaining 78% of all page loads are subjected to tracking.

Figures 1a and 1b also show that the proportion of unique domains (sites) with large numbers of *trackers* is less than the number of pages loaded with this many trackers, as expected given the positive correlation between tracker's occurrence and popularity of the sites.

3.2 Trackers Reach

Naturally not all *potential* trackers belong to the same organization so it would be incorrect to conclude that a single organization can track users across 78% of their browsing. That would only be the case if trackers were to share data, note that some trackers share data via exchanges, e.g. bidding for users.

To assess approximately the reach of different organizations we use *Disconnect* [7] blocklist, which groups its 2000 trackers by organization. Figure 2 depicts the top 20 organizations ranked by their aggregated tracker's reach.

All trackers owned by Google combined (according to *Disconnect*'s blocklist) are present on 62% of page loads. If we restrict the analysis to those page loads with *unsafe* data elements, Google's reach lowers to 42%. This means 42% of the pages loaded by 200,000 Germans for a 7 days period produced a request from the user's browser to a tracker controlled by Google and that request contained data that is or could be used as a user identifier.

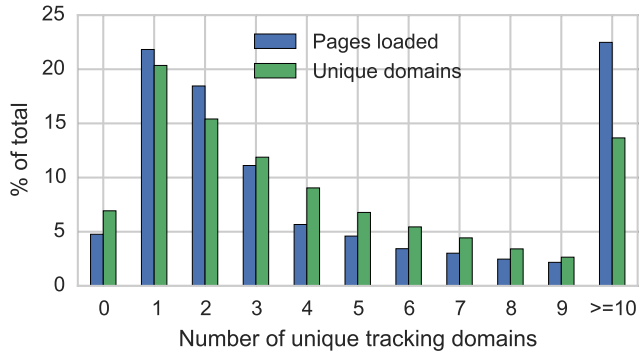
Criteo, an organization that provides ad-retargeting, is in the 4th position on the list, with a reach of 9% of the page loads. Note that 9% might look like a modest number, especially when comparing it to *Google* or *Facebook* but it is not. 9% translates to roughly 1.2 million page loads, about 1.1 tracking signals per user per day.

The results collected from our large scale study are conclusive: tracking users on the Web is wide-spread phenomena not limited to popular or heavy monetized sites.

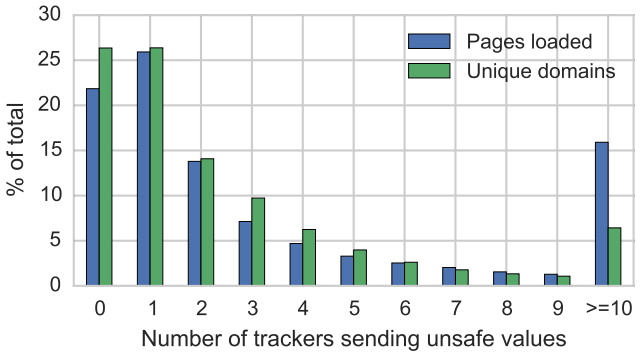
[4]Between 28/09/15 and 04/10/15

[5]Kontaxis and Chew [19] included longitudinal data from users running Firefox nightly (140K) but only 0.5% of those users had the Firefox anti-tracking enabled.

[6]A more formal definition is provided in Sect. 4.1



(a) Requests to *potential* trackers.



(b) Requests to *potential* trackers with *unsafe* data detected.

Figure 1: Distribution of number of requests to *potential* trackers from 21 million page loads from 200,000 users. The 0 column means that zero trackers were present on the page load, thus no tracking.

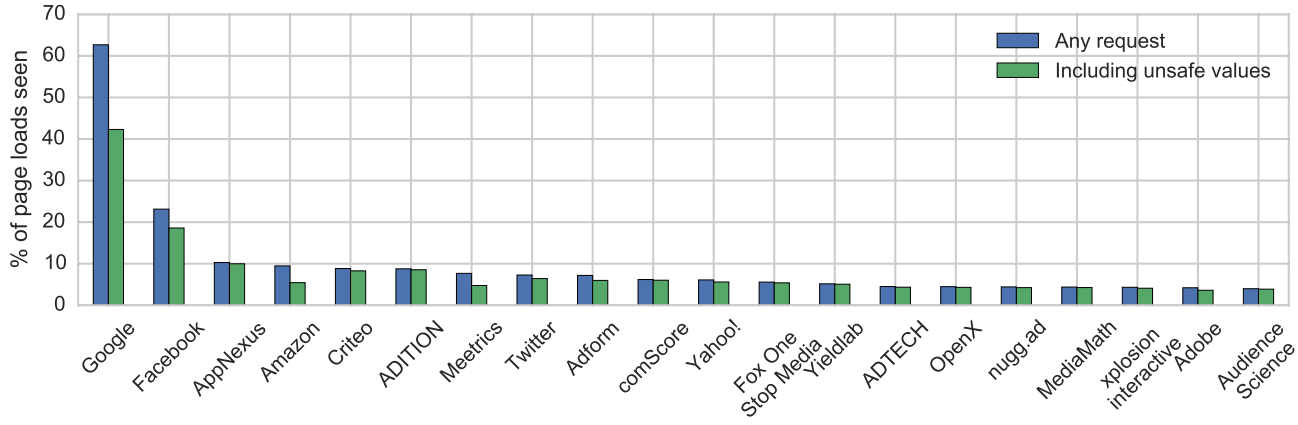


Figure 2: Top 20 organizations by combined tracker's reach. The ownership of a tracker is based on Disconnect's blocklist [7]. The first column accounts for the percentage of page loads in which a request to the *potential* tracker is issued by the user's browser. The second column is when the request also contains *unsafe* data.

People who are not concerned about privacy might consider that there is no real risk if an organization such as *Google* can track which articles they read on *Engadget.com* or *Quora.com* since they can be perceived as neutral with regard to privacy. But this is not the full picture, tracking is prevalent on virtually all sites. Trackers owned by *Google* are also present on pages on *AshleyMadison.com*, videos on *PornHub*, posts about diseases of *Gutefrage.net* and a long list composed of tens of thousands of sites. Organizations able to track users across thousands of sites have the ability to infer aspects like religious beliefs, trade union membership, sexual orientation, etc. Handling this kind of data requires stronger requirements and generally the users' consent according to German law [5].

4. DESIGN AND IMPLEMENTATION

As we already stated throughout this paper, our approach is not based on detecting 3rd party trackers but rather on dynamically identifying data elements that are not *safe* with regard to privacy. In this Section we will formally define concepts like *safe*, *unsafe* and *potential* trackers as well as show how they can be implemented in a system.

4.1 Determining Safeness

The typical approach is based on trying to make sense of what 3rd parties are trying to achieve in order to classify them as trackers or not.

In this paper we advocate for a different approach to the problem, related to k-Anonymity [38]. Rather than scanning for data elements that could be used as a user identifier (*uid*) by analyzing its semantics, origin or method of creation we rely on a simple fact: if a data element was intended by a 3rd party be used as a *uid* then we should expect that only and only that user is sending that particular data element.

Therefore, our rationale is simple, data elements that are only and always sent by a small number of users must be considered *unsafe* with regard to privacy. Such elements are dangerous since they are intended to serve as user identifiers (*uid*) or, in the case there was no intention, they could be used as such. On the other hand, data elements, no matter how obscure or suspicious they look, that have been sent by many independent users can be considered *safe* because the

data cannot be effectively related to any particular user (in upcoming Sect. 4.1.1 we will present some caveats of this claim).

We assess *safeness* as follows: for every request that user's u browser makes to a 3rd party, the *QueryString* component of the request is parsed to obtain a list of tuples where each tuple t takes this form,

$$t = \langle u, s, d, k, v \rangle \tag{1}$$

where u is the user, s is the source domain (the domain of the page the user visited), d is the 3rd party domain, k is a parameter and v is the parameter value. To mitigate known caveats of k-Anonymity on high-dimensionality data [27, 2, 28] we do not limit k and v to be the explicit parameter/value in the *QueryString* (see Sect. 4.1.1 for further details). For clarity, let us assume that users share the tuple t with a centralized service hosted by Cliqz without worrying about privacy (see Sect. 4.2) or efficiency (see Sect. 4.3).

The first thing to determine is whether the 3rd party domain has a connectivity behavior that can be associated with trackers. We classify a 3rd party domain as ***potential tracker*** if the 3rd party domain d is (1) present in more than 20 different source domains s in a 7 days period, (2) has been seen at least 5000 times in this period, and (3) a proportion of requests have been observed attempting canvas fingerprint, sending private browser data, sending cookie values, or frequently sending *QueryString* data[7].

Note that *potential* trackers can be seen as some sort of a blocklist, however, due to the mixed behavior of trackers we cannot base our privacy intervention solely on it. In the case that a 3rd party is on our list of *potential* trackers we need to determine their ***safeness level***. We will consider any data element that has been seen at least k_g times by different users in a t_g period *safe*, formally,

$$\text{safe} \iff |\langle u = *, d, v \rangle| \geq k_g \text{ for } t_g \tag{2}$$

the values of key and source domains (k and s, respectively) are aggregated, therefore not shown in the equation. What we are looking for is the cardinality of the set of unique users u for which a given value v was sent (attempted) to a given *potential* tracker d. In our current configuration, k_g is set to 10 and t_g is set to 7 days, i.e. a data element v sent to d will be *safe* only when it has been seen by at least 10 different users in a period of 7 days. If that condition is not met the data v should be altered (or blocked) to prevent the *potential* tracker from receiving it.

The default *safeness* level for data that has never been seen is, consequently, *unsafe*. This implies that all new data elements collected by a 3rd party that are bound to become *safe* must undergo a *transient state*, in which they will be considered *unsafe*. That is the only way by which one can be sure that the data is not tied to any particular user.

The *transient state* can be split in two parts: 1) the time it takes for the value to reach the *safeness quorum*, determined by k_g and by the number of requests containing that value. And, 2) the *propagation lag*, which is the time it takes from when *safeness quorum* is reached until users learn about it. We defer the analysis of the impact of it to Sect. 5.4, but we can anticipate that it is very small.

[7]The proportions tested for each behavior are 5%, 5%, 1% and 50% respectively.

When the *safeness quorum* is reached the data element is added to a set G_{wl} that acts as the whitelist of *safe* values. This set is shared to all browsers by an updating mechanism described in Sect 4.3.1 so that local browser can decide locally whether to allow or prevent a value to be sent to a *potential* tracker. Relying only on G_{wl} poses a couple of problems that need to be addressed, one is size, which is quite critical since G_{wl} needs to be transferred to all browsers. The other is the *transient state* itself, which has to be minimized.

To address the shortcomings from G_{wl} we introduce another set that is built locally in each of the user's browser called L_{wl}. This set is built following Eq. 3 which does not require global information,

$$\text{safe} \iff |\langle d, k, v = * \rangle| \geq k_l \text{ for } t_l \tag{3}$$

if the same key k for a *potential* tracker d has been seen to hold more than k_l different values v for a period of t_l the whole combination of $d-k$ is considered *safe*. In the current configuration, k_l is 3 and t_l is 7 days. This means that the user has seen at least 3 different values for the same $d-k$ combination (key and *potential* tracker respectively). This helps to decreases the *transient state* for a wide variety of cases, e.g. a parameter from a 3rd party encoding a *product-id* or a *timestamp*. In such cases the combination $d-k$ will be added to L_{wl} and also considered *safe*, not by value, but by key, i.e. any value for the key k and *potential* tracker d combination will be allowed.

If the key is considered *safe* it implies that all values of the key are also *safe*, and consequently, the key can be used to prune the global set G_{wl}. Instead of holding every possible *safe* value of $d-k-*$ it will only keep $d-k$ that subsumes them. The pruning is performed locally but it can also be performed globally on the G_{wl} if certain conditions are met. When a browser adds a new $d-k$ combination to L_{wl} it will be sent to the Cliqz service. If more than 10 different people per hour report that a $d-k$ is *safe* it will be added to G_{wl} with an expiration time of two days, this will cause pruning of G_{wl} with the consequent reduction of size.

4.1.1 Robustness against Attacks

It is important to stress that our method has some angles that an adversary might attack to subvert the *safeness* evaluation. This would be true even in the case were the inner working of the system were not publicly available. In any case, vectors of attack can be prevented once known.

An adversary might try to exploit the collaborative nature of our system, for instance by creating instrumented users that collude to define a combination $d-k$ *safe* when it is not, thus exploiting Eq.3 pruning on G_{wl}. This can be prevented using anomaly detection and selectively testing the validity of the client's claims on the server side.

An adversary might also try to attack known vulnerabilities of k-Anonymity which plays an important role in our *safeness quorum*. k-Anonymity suffers from issues on high-dimensionality data [27, 2, 28], one can take a combination of multiple variables that are not unique but once combined they become unique. For instance, an adversary might break a 32 bits *uid* parameter into 4 different parameters encoding 8 bits each, the values then would most likely pass the *safeness quorum* since they can only take 256 different values across the whole population. We counter such attacks, departing from vanilla k-Anonymity, because our list of tuples

(Eq. 1) is not limited to explicit parameters of the data, we also create artificial parameters where values are combined and split. Following the attack example, if two parameters encoding partial *uid* were to be combined that would yield a virtual parameters encoding 16 bits (65K different values) likely to fail on the *safeness quorum*.

We have outlined two potential attacks and their countermeasures but there are more vulnerabilities, including unknown ones for which no countermeasure can be devised until discovered. Our method is not immune to attacks—a "smart" tracker can try to subvert our system to continue tracking users on the Web. We are not getting rid of the *arms-race* between trackers and people working on privacy protection. It will continue. But thanks to our system trackers will have a much harder time; it is no longer enough to hide or obfuscate the origin of a particular data element, trackers will also have to "convince" our system that enough members of the community have seen it.

4.2 Private Data Sharing

In the previous section we saw that we needed to collect tuples $\langle u, s, d, k, v\rangle$ to determine *safeness*. These tuples, however, constitute a privacy breach since they contain the user identifier u and the source domain s of the page visited, which is precisely the information that trackers should not collect. Naturally, we do not collect that information either.

To maintain privacy we need to make sure that no user identifier u is ever sent to *Cliqz*. The user browser will keep an in-memory counter of the signatures of seen tuples, bucketed by hour. The first time the tuple is seen in one hour period the increment will yield a 1, then and only then tuple (with u removed) will be sent to *Cliqz*. Following this scheme, users will only send the same signal (tuple) once per hour if seen. As a consequence, the server can assume that a repeated signal in the same hour interval comes from two different users, without having to rely on the explicit user identifier u.

Preventing the user identifier u to reach Cliqz's servers is not enough, the tuple still contains sensitive information. By design, our system only requires comparisons between strings, so the real values are not needed and can be obfuscated by a one way function. All values from the tuple t will be hashed using MD5. Additionally, s and d can also be truncated to keep only the first 16 chars of the hash to allow for plausible deniability at the expenses of a very small collision probability. We are aware that a brute-force dictionary attack to reverse the MD5 is possible but quite unfeasible given the wide range of values that k and v can take.

The same applies to the tuple $\langle d, k\rangle$ from the user's L_{wl} set. Once per hour, if the browser is active, the full list of items contained in L_{wl} will be sent to Cliqz.

Data is sent to *Cliqz* using the *Human Web* framework [22], which provides communication anonymity by means of a proxy relay. Furthermore, we must avoid the possibility to create sessions on our server side. To achieve this the browser will send each signal on a new request (to avoid grouping) and the request will be sent at a random time since creation (to avoid temporal correlations).

4.3 Implementation

Let us now present the implementation of the system in the browser. Every request to a 3rd party will be suspended until the checks below are resolved:

1. 3rd party cookies will always be blocked, regardless if the 3rd party is a *potential* tracker or not.

2. if a *GET* request, we will build the list of tuples T from the QueryString data following Eq. 1. Tuples whose value v is shorter than 12 characters (or 8 in the case the value is not a word) are excluded. For each t in T we will calculate its *safeness* level,

$$\begin{aligned} &concat(H_t(d), H(k)) \in L_{wl} \vee \\ &concat(H_t(d), H(k)) \in G_{wl} \vee \\ &concat(H_t(d), H(v)) \in G_{wl} \implies \text{safe} \end{aligned} \tag{4}$$

   by building a string from the concatenation of d, k and v hashes, and doing a membership tests against the sets L_{wl} (implemented as a Dictionary) and G_{wl} (implemented as a Bloom Filter to minimize size, with a false positive rate of 0.1%). If not *safe* the data element will be blocked, either by: replacing it by an empty string, replacing it by a placeholder string or by shuffling the value.

3. if a *POST* request. Data elements in the *POST* will be blocked by a more conservative criteria. If they match a value of the *Cookie*, or if they match a value from a call to *browser-info* like *plugins*, *build-id*, *oscpu*, or if values are also present in the *QueryString* and were not *safe* according to the previous point.

4. all tuples in T are queued for the off-line component of the anti-tracking system.

5. check that exceptions do not apply (see Sect. 4.3.2).

This real-time checking adds between 1 to 12 ms to each 3rd party request depending on the number of parameters. It is not a negligible amount but note that these calls are usually asynchronous to the page load.

The off-line component responsible for building the G_{wl} and L_{wl} sets is run as an independent process decoupled from the real-time requests. The memory footprint can be broken down to 390 KB for G_{wl} once implemented as Bloom Filter. G_{wl} contains on average about 100K elements (28% of them are key-based and the rest are values). On the other hand, L_{wl} and the dictionary used to keep counters of the tuples' signature bucketed by hour account for less than 20KB.

The bandwidth overhead details are as follows, besides telemetry, all the signals from the user's browser to Cliqz servers amount to 90 KB per day on average. This figure does not include maintaining consistency between the global G_{wl} that the copy running on the users' browser, which will be dealt in the next section.

4.3.1 Maintaining G_{wl} Consistency

The global whitelist set G_{wl} is built in a centralized way, using the anonymous signals contributed by all users running the anti-tracking system. G_{wl} is highly dynamic, keys and values are added as per Eq. 3 and 2 and removed according to their expiration times. The assessment for *safeness* however is based on the local copy of G_{wl} in each user's browser as per Eq. 4. Thus maintaining consistency among the global and the local copy of G_{wl} is a key factor in our system.

We rely on an eventual consistency model; changes on the global set will be reflected eventually to the local one. However the time it takes for a change to propagate is important, and needs to be bound. We aim to minimize the *propagation lag*. That is values that are declared *safe* on the global G_{wl} but that local copies are still treating as *unsafe* and, therefore, blocking it. However, a fast propagation time also involves higher bandwidth costs for the browsers.

After some experimentation we settled for a setup by which Cliqz publishes a brand new G_{wl} set daily as a major version (typically 350 KB) and every 10 minutes incremental additions are published as revisions to that version (typically 1.5 KB). This version/revision setup is due to the particularities of Bloom filters, which allow for very efficient merge operations (N sets can be merged using an bit-wise OR operation) but does not allow set removals operations. Therefore, we cannot provide a fully incremental solution and daily checkpoints, i.e. the major version, are required to accommodate the expiration of keys and values.

With the current setup, a browser that is always online will have a to download about 566 KB per day to have a worst-case *propagation lag* of 10 minutes.

4.3.2 Exceptions to the Rule

There are certain cases in which data elements that are *unsafe* are not blocked for the sake of minimizing breaking the site's functionality or appearance. As mentioned in Sect. 2.2 site breakage can impact negatively privacy coverage as the result of custom rules an exceptions added by the users trying to make the sites work as expected. This particular issue will further analyzed in Sect. 5.2 where we empirically measure site breakage.

Another exception revolves around 3rd party *DOM* elements, for instance sites with the Google *+1* button, or sites using *Disqus* to power its commenting system, which in turn allows for *OAuth* login through *Facebook* and *Twitter*. When a user explicitly interacts with a 3rd party *DOM* element the anti-tracking validation will be suspended for any 1st or 3rd party domain that can be attributed to trigger as a consequence of that event. Unfortunately perfect attribution is not always possible, due to complex work-flows involving callbacks, redirects and nested calls. So we must rely on heuristics to determine attribution. A case by case analysis seems to indicate that our attribution heuristics do not suffer from false positives but we cannot rule out the possibility that they exists. In the case of a false positive attribution we might allow data that did not undergo our *safeness* evaluation reach a tracker.

5. SYSTEM EVALUATION

In this section we will proceed to the evaluation of the anti-tracking system in different dimensions.

5.1 Protection Coverage

The first thing to evaluate is the protection coverage offered by our system compared to related work. We chose *Disconnect*, since they have published their blocklist, and Kontaxis and Chew [19] (henceforth *KC*), which is based on a hand-picked subset of 1500 domains of Disconnect's blocklist selected to minimize site breakage[8]. Although we would have liked to include *Adblock Plus* [31] in the measurement it was not technically possible[9].

[8]Since we do not know which subset was picked we evaluated 1000 random subsets based on Disconnect blacklist and kept the one with better protection coverage.

Table 1: Privacy coverage contingency tables. For each configuration the contingency table shows the percentual breakdown of request to 3rd parties according to whether they are in a blocklist (denoted b) or not and whether they contain *unsafe* data elements or not (*unsafe* denoted $\neg s$). Note that b for the case of *Cliqz* means that is on the list of *potential* trackers, not necessarily blocked. Whereas b for *Disconnect* and *KC* means that is on the blocklist, therefore, blocked.

		All Pages		Alexa top 200 US domains	
		b	$\neg b$	b	$\neg b$
Cliqz	s	31.8	12.5	56.1	10.4
	$\neg s$	**51.7**	4.0	**31.5**	2.0
Disconnect	s	**25.7**	18.6	46.2	20.3
	$\neg s$	**40.4**	**15.3**	**27.6**	**5.9**
KC: Kontaxis & Chew (est.)	s	**10.7**	33.6	9.3	57.3
	$\neg s$	**25.7**	**30.0**	**19.6**	**13.6**

We choose an direct empirical evaluation rather than a side-to-side comparison on a limited set of sites like [33]. Such experiments provide more detail on the inner workings of the systems analyzed but they cannot quantify the privacy protection in a real-world scenario. Since we have the luxury of data on 350K unique sites visited by 200K users we believe that a direct evaluation is more relevant.

Table 1 depicts the requests contingency tables for the 3 anti-tracking methods in two different scenarios, requests to any page, and requests to pages from the top-200 visited sites in the US, according to Alexa. The contingency table splits the requests according to two different attributes: 1) s and $\neg s$, whether the requests contained only *safe* data elements or not. And 2) b and $\neg b$ whether the 3rd party of the request was in the blocklist of the method. Each set of four values in the table add up to 100% of the requests.

Let us focus first on the all pages scenario. *Cliqz* anti-tracking (our system) makes an intervention to protect privacy on 51.7% of the requests, when the 3rd party is a *potential* tracker and the request contains an *unsafe* data element ($b \wedge \neg s$). *Disconnect* makes an intervention (blocks) whenever the 3rd party is on its blocklist, which happens for 66.1% of the requests (40.4% from $b \wedge \neg s$ + 25.7% from $b \wedge s$). This seems to indicate higher coverage from *Disconnect*, but 25.7% of the requests blocked contained no *unsafe* data elements at all. If we calculate the ratio of blocked and *safe* by all blocked we will see that 38.8% of the blocked requests by *Disconnect* are false positives according to our *safeness* measure.

Blocking these requests is unnecessary, it does not increase the privacy coverage and can result on an increase of site breakage. *KC* reported that they used only a curated subset of the full *Disconnect* blocklist to minimize site breakage, our estimated results on *KC* show that unnecessary blocking of requests drops to 29.4%, thus minimizing the probability of site breakage due to unnecessary blocking.

[9]Adblock Plus blocklist contains a large number of regular expressions which cannot be evaluated with the data available to us. For privacy reasons the 3rd party domain for each requests is hashed, see Sect. 4.2

Less aggressive blocking has the side-effect of allowing requests that should have been blocked. For *KC*, using the best estimated subset of *Disconnect* blocklist, we observe that 30.0% of all requests were not blocked and contained *unsafe* elements ($\neg b \wedge \neg s$), thus we can consider these as *misses* in privacy protection. For the full blocklist of *Disconnect* the *misses* percentage drops to 15.3%. According to our *safeness* criteria, both systems allow a fair amount of requests that contain privacy sensitive data. Note that for *Cliqz* ($\neg b \wedge \neg s$) is not zero but 4%, this accounts for the number of requests with *unsafe* elements send to 3rd parties not in our list of *potential* trackers. These requests are allowed regardless of the *safeness* level because we consider 3rd parties that do not meet the criteria for *potential* trackers are not dangerous, due to lack of reach (see Sect. 4.1).

Taking this into consideration we should revisit the percentage of *misses* for *Disconnect* and *KC* to exclude requests not present in our list of *potential* trackers, i.e. not dangerous because of limited reach. The revisited *misses* figures are 12.3% for *Disconnect* and 25.4% for *KC*, still very high, showing that both systems let a fair amount of requests to *potential* trackers containing *unsafe* data pass through.

The same analysis can be performed by the requests to the top-200 sites according to Alexa. Results are consistent, and, as one might expect, popular sites have better protection coverage. Still, the coverage offered by *Disconnect* is not total, 5.9% of the requests still contain *unsafe* data elements, 4.7% in the case we exclude requests not in our list of *potential* trackers.

The better performance in terms of coverage of our system mostly arises from the ability to block not at the coarse-grained level of the 3rd party request, but rather on the requests' individual data elements.

Figure 3 depicts the cumulative probability that a tracker present both in *Disconnect* blocklist and in our list of *potential* trackers contain an *unsafe* data element. We can observe that as many as 78% trackers have a mixed behavior, sometimes sending *unsafe* data sometimes not. Thus a heavy-side function like a binary classification blocklist will not be a good fit. Anti-trackers already realized that blocklists need a finer-grain resolution to deal with the mixed behaviour of trackers. Allowing exceptions and/or rules based on full URL or even regular expressions, like *Adblock Plus* are attempts to increase the degrees of freedom of the blocklist classification. However, it is highly unlikely that a case-based approach will be as scalable and as effective as looking at the *safeness* level of individual data elements.

5.2 Measuring Site Breakage

We now turn to evaluate the site breakage produced by our system. We already mentioned multiple times that the trade-off between breakage and protection coverage is not trivial. Increasing protection at the expenses of increasing site breakage can lead to a decrease of protection. If site breakage is high users will eventually take action either by removing the tracking protection or by creating custom rules or too general exceptions that are bound to decrease their protection. User intervention, while an interesting feature for power-users, can be considered harmful if abused.

Measuring site breakage on 350K sites is something that cannot be achieved with typical QA. *KC* proposed an indirect measurement, monitoring how many times users click on the button to temporally disable anti-tracking. This has some limitations: 1) it assumes that users are likely to interact with a new UI element that they are not familiar with. And 2) not all systems under study have such button, and if the button exists, we do not have access to the signal. We decided to use an alternative measure based on the *page reload rate*. The underlying assumption is that when users experience a site with broken functionality or appearance they will attempt to reload the page, which is a more natural action on a browser.

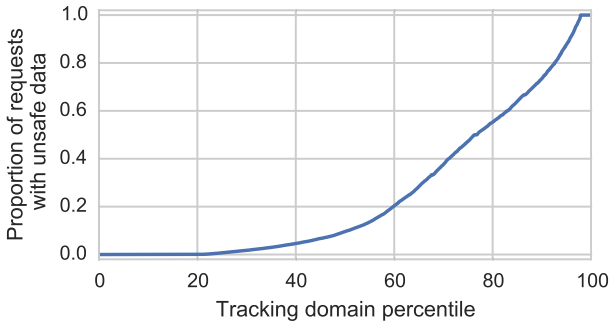


Figure 3: CDF of the number of requests with observed *unsafe* data-elements by 3rd party domains contained both in Disconnect blocklist and on Cliqz list of *potential* trackers. Both blocklist have ∼2000 domains each, intersection is 447.

Table 2: Page reload rates

	reload rate	% increase
Baseline (No Anti-Tracker)	0.00101	–
Cliqz	0.00104	4%
Adblock Plus	0.00110	10%
Cliqz acting as Blocklist	0.00125	25%

Table 2 shows the observed page reload rates based on an A/B test of our users running different anti-tracking browser extensions. The first group is a control group, users that have no anti-tracking present. The second group are users running *Adblock Plus*. From our data we see that 33% of requests are from browsers running *Adblock Plus*. Firefox 3rd Party Cookie Blocking [24], seen on 2% of requests, and other anti-tracking extensions, including *Disconnect*, had too few users for reload rate to be measured with statistical significance. The last group of users are running the *Cliqz* anti-tracking, as presented in this paper. Note that these groups are independent, but thanks to the large population in each group we can assume user behavior is consistent.

The baseline page reload rate is 0.001, approximately 1 reload for every 1000 pages visited. For users running *Adblock Plus* the reload rate increased almost 10% from the baseline, so it is plausible to assume that the increase of the page reload rate can be attributed to site breakage caused by it. For *Adblock Plus*, some breakage could be caused by its DOM manipulation phase, which is independent from the anti-tracking dimension. Note that users running *Adblock Plus* might have created custom rules and exceptions to decrease the site breakage they experience, such users are included in the results[10].

[10]For privacy reasons we cannot collect the users' Adblock configuration to assess the level of customization

Table 3: Origin of *unsafe* data. *Cookie* corresponds to the transmission of a cookie value in another part of the request (i.e. an attempted circumvention of cookie blocking). 2) *Javascript fingerprinting* refers to an attempt to send private values generated from javascript functions (e.g. browser-info, names of installed plugins, build-id, oscpu and other and OS information). 3) *Canvas fingerprinting* accounts for a method of generating *unsafe* data using the HTML5 canvas element. 4) *Unknown* accounts for the rest.

unsafe data creation method	Proportion of *unsafe* requests
Cookie value	58%
Javascript fingerprinting	15%
Canvas fingerprinting	0.5%
Unknown	40%

For users running *Cliqz* the page reload rate increased 5% over the baseline, thus we can conclude that our anti-tracker still produces undesired site breakage. However, it is less than the community-optimized *Adblock Plus*, without having to resort to user intervention of any kind.

We also include the page reload rate for a modified version of our system (*Cliqz*) that acted as a blocklist. Any request to a 3rd party in the list of *potential* trackers was blocked. In such scenario 83.5% of the requests would be blocked (as per Table 1) instead of 51.7%. This aggressive blocking did not increase privacy coverage—blocking requests to *potential* trackers if all data is *safe* has no benefit and it should increase the chances of site breakage. The observed page reload rate supports this claim: during the period[11] that *Cliqz* acted as a blocklist there was a 25% page reload rate increase over the baseline.

5.3 Unsafe data origins

The common approach to finding trackers involves introspection of the requests sent to 3rd parties to find common patterns to generate *uid*s. We wanted to quantify how many *unsafe* requests detected by our system could have detected solely by their creation method.

Results summarized in Table 3 show that 58% of *unsafe* requests are due to attempts to circumvent Cookie protection, for instance, sending cookie values on the *QueryString*. Fingerprinting based on exploiting javascript, including strict Canvas fingerprinting, amounts for 15.5% of the observed *unsafe* requests. Finally, for 40% of the *unsafe* requests we cannot determine the origin of the *unsafe* data elements, illustrating the difficulty of data inspection.

5.4 Measuring Transient State

Finally we measure the effect of the *transient state*, produced by the time it takes to reach the *safeness quorum* plus the *propagation lag*. During this time data elements that are truly *safe* are treated as *unsafe* by our system, i.e. they are temporarily miss-classified.

Rather than measuring time, which has factors that depend on externalities like popularity (to reach *safeness quorum*) we focus on measuring the number of occurrences of the phenomena. To that end we force the users' browser to keep a record of data elements that were declared *unsafe* for few days and reevaluate their *safeness* level periodically to see if its classification changes to *safe*. The frequency of such cases gives us the miss-classification rate.

Results show that approximately 650M data elements are analyzed per day. 0.8% of them are declared *unsafe*. Only 0.07% of the data elements declared *unsafe* were caused by a miss-classification due to the *transient state*. Therefore, we can safely conclude that the effect exists but it is almost negligible.

[11] between 09/09/2015 and 16/09/2015

6. CONCLUSIONS

In this paper we proposed a novel approach to prevent online tracking. Thanks to a large scale empirical study, involving 200,000 users in Germany, we showed the protection coverage achieved is better than other systems, such as *Disconnect*, which failed to account for at least 12.3% of the requests that contain privacy sensitive data, despite having aggressive blocking rules. We also showed that the extra privacy protection does not come at the expense of breaking sites' functionality or appearance, the disturbances to the user can be measured to be half of those caused by mature and highly optimized systems such as *Adblock Plus*. All this with a system that is purely algorithmic, and requires no user intervention at any time.

The novelty of our approach is its departure from the traditional blocklist approach. Rather than trying to identify 3rd parties as trackers we increase the classification resolution by basing our decision on the data elements of the requests. We have built a system that efficiently, and respecting the privacy of the users, allows them to collaborate to collectively determine if the data elements present in a request are *safe* or not. Only those data elements that have been seen by enough independent users will be considered *safe*. Thus, data that is intended by trackers to serve as user identifiers (*uids*), or that is unique enough that it could be used as such, are filtered out, protecting the user's privacy. The proposed model is inspired by k-Anonymity, although it departs from it in several ways to cover for some known vulnerabilities.

Our large scale empirical study based on more than 21 million pages visited on 350,000 different sites by 200,000 users over a week allows us to evaluate the prevalence of trackers in the wild. We have shown that trackers are present is more than 95% of the pages visited by Germans, and as little as 22% of the page loads do not attempt to transfer data that is considered *unsafe* with regard to privacy. We also provided evidence of the reach of the organizations behind trackers, establishing a ranking.

Despite the very positive empirical results obtained by our system we cannot call victory against online tracking. The *arms-race* will continue, the system presented in this paper is making the life of trackers much more difficult, but there are still ways in which trackers can continue to monitor the user, even with our system in place. Future work will focus on extending the protection against some attacks to subvert the system, as well as improving the overall efficiency of it.

7. ACKNOWLEDGMENTS

We would like to thank all *Cliqz* users who by using *Cliqz* contributed with the anonymous data required to develop, test, and eventually operate, the tracking protection system presented in this paper.

8. REFERENCES

[1] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[2] C. C. Aggarwal. On k-anonymity and the curse of dimensionality. In *Proceedings of the 31st VLDB Conference,*. ACM, 2005.

[3] B. V. Alsenoy, V. Verdoodt, R. Heyman, J. Ausloos, E. Wauters, and G. Acar. From social media service to advertising network. `http://www.law.kuleuven.be/citip/en/news/item/facebooks-revised-policies-and-terms-v1-2.pdf`, 2015.

[4] R. Baloch. Facebook cookie stealing and session hijacking. `http://www.rafayhackingarticles.net/2011/07/facebook-cookie-stealing-and-session.html`.

[5] Bundesbeauftragte für den Datenschutz und die Informationsfreiheit. Bundesdatenschutzgesetz (BDSG), 3(9). `https://www.bfdi.bund.de/bfdi_wiki/index.php/3_BDSG_Kommentar_Absatz_9_Beispiele`.

[6] Department of Homeland Security. Increase in insider threat cases highlight significant risks to business networks and proprietary information. `http://www.ic3.gov/media/2014/140923.aspx`, 2014.

[7] Disconnect. Disconnect. `https://disconnect.me/`.

[8] P. Eckersley. How unique is your web browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*. Springer-Verlag, 2010.

[9] Electronic Frontier Foundation. Privacy badger. `https://www.eff.org/privacybadger`.

[10] European Parliament and of the Council. Directive 2002/58/EC of 12 july 2002 concerning the processing of personal data and the protection of privacy in the electronic communications sector. *Official Journal L 201 , 31/07/2002 P. 0037 - 0047*, 2002.

[11] T. K. Franziska Roesner and D. Wetherall. Detecting and defending against third-party tracking on the web. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012.

[12] B. Gaylor. Do not track: a personalized documentary series about privacy and the web economy. `https://donottrack-doc.com/en/`, 2014.

[13] Ghostery. Ghostery. `https://ghostery.com/`.

[14] G. Greenwald and E. MacAskill. NSA Prism program taps in to user data of Apple, Google and others. `http://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data`.

[15] S. Guha, B. Cheng, and P. Francis. Privad: Practical privacy in online advertising. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

[16] C. Hoback. Terms and conditions may apply. `http://tacma.net/`, 2013.

[17] Internet Advertising Bureau. 2013 internet ad revenues soar to $42.8 billion. `http://www.iab.net/about_the_iab/recent_press_releases/press_release_archive/press_release/pr-041014`, 2014.

[18] H. P. Jason Bau, Jonathan Mayer and J. C. Mitchell. A promising direction for web tracking countermeasures. In *Proceedings of Web 2.0 Security and Privacy (W2SP)*. IEEE Computer Society, 2013.

[19] G. Kontaxis and M. Chew. Tracking protection in firefox for privacy and performance. In *Proceedings of Web 2.0 Security and Privacy (W2SP)*. IEEE Computer Society, 2015.

[20] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining,*, pages 1245–1254, 2009.

[21] M. Madden. Public perceptions of privacy and security in the post-snowden era. `http://www.pewinternet.org/2014/11/12/publicprivacy-perceptions/`.

[22] K. Modi and J. M. Pujol. Collecting user's data in a socially-responsible manner. In *European Big Data Conference*. Linux Foundation, 2015.

[23] K. Mowery and H. Shacham. Pixel perfect: Fingerprinting canvas in html5. In *Proceedings of W2SP 2012*. IEEE Computer Society, 2012.

[24] Mozilla Corporation. Disable third-party cookies in firefox to stop some types of tracking by advertisers. `https://support.mozilla.org/en-US/kb/disable-third-party-cookies`.

[25] Mozilla Corporation. New experimental private browsing and add-ons features ready for pre-beta testing in firefox. `https://blog.mozilla.org/futurereleases/2015/08/14/new-experimental-private-browsing-and-add-ons-features-ready-for-pre-beta-testing-in-firefox/`.

[26] E. Nakashima. Verizon says it turned over data without court orders. `http://www.washingtonpost.com/wp-dyn/content/article/2007/10/15/AR2007101501857_pf.html`.

[27] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. *IEEE Symposium on Security and Privacy*, 10(5):111–125, 2008.

[28] T. L. Ninghui Li and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. *IEEE 23rd International Conference on Data Engineering*, pages 106–115, 2007.

[29] NoScript. NoScript. `https://noscript.net/`.

[30] PageFair and Adobe. Adblocking goes mainstream. `http://downloads.pagefair.com/reports/adblocking_goes_mainstream_2014_report.pdf`.

[31] W. Palant. Adblock plus. `https://adblockplus.org/`.

[32] A. Peterson. Bankrupt radioshack wants to sell off user data. but the bigger risk is if a facebook or google goes bust. Washington Post, March 26th 2015.

[33] Raymond. 10 ad blocking extensions tested for best performance. `https://www.raymond.cc/blog/10-ad-blocking-extensions-tested-for-best-performance/view-all/`.

[34] C. Riederer, V. Erramilli, A. Chaintreau, B. Krishnamurthy, and P. Rodriguez. For sale : your data: by : you. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks (HotNets)*, 2010.

[35] T. Sharp. Right to privacy: Constitutional rights and privacy laws. http://www.livescience.com/37398-right-to-privacy.html, 2013.

[36] S. Sheng, B. Wardman, G. Warner, L. F. Cranor, and J. Hong. An empirical analysis of phishing blacklists. In *Proceedings of 6th Conference on Email and Anti-Spam (CEAS)*, 2009.

[37] Stanford Encyclopedia of Philosophy. The constitutional right to privacy. http://plato.stanford.edu/entries/privacy/#ConRigPri.

[38] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty*, 10(5):557–570, 2002.

[39] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *Proceedings of the 17th Annual Network and Distributed System Security Security Symposium*. The Internet Society, 2010.