Data Collection without Privacy Side-Effects

Konark Modi
Cliqz
konark@cliqz.com

Josep M. Pujol
Cliqz
josep@cliqz.com

ABSTRACT

The standard approach to collect users' activity data on the Web relies on server-side processing. This approach requires the presence of user-identifiers in order to aggregate data in sessions, which leads to tracking. Server-side aggregation is bound to produce side-effects because the scope of sessions cannot be safely limited to a particular use-case. We provide several examples of such side-effects.

To preserve privacy we propose an alternative approach based on client-side aggregation, where user-identifiers are not needed becasue sessions only exist on the client-side (i.e. the user's browser). We demonstrate the feasibility of this approach by providing an implementation of a tracking agent – *green-tracker* – able to gather the data needed to power a service functionaly equivalent to Google Analytics.

1. INTRODUCTION

The data processing aspect of *Big Data* has received a lot of attention; resulting in many great technologies like Hadoop, Spark, GFS as well as advances in networking and storage, etc. On the other hand, not so much attention has been paid to where that *Big Data* comes from.

The industry *modus operandi* can be described as *collect-all-you-can*, and this behavior is not only accepted but encouraged. If one considers data as the energy to power services, then one will naturally seek to hoard it. This approach to data collection, however is dangerous especially when data involves humans. For cases like Large Hadron Collider at CERN there is no discussion on whether more data is better. Particle collision data is unlikely to have side-effects, and even if there were any, particles are not likely to complain. However, when the subject of data collection involves human actions on the Web, side-effects are something to be considered seriously.

What is a side-effect? In simple terms, a piece of knowledge that can be learned from data analysis which was neither intended nor expected, and that poses a risk to the privacy of the subjects of the data collection.

2. SIDE-EFFECTS ARE UNAVOIDABLE

Practitioners in data collection have no desire for such side-effects. Unfortunately, the current approach to data collection, where aggregation only takes place on the back-end of the collector, makes them unavoidable.

Let us illustrate this point by analyzing one of the biggest data collectors in the Web: Google Analytics (henceforth GA). According to [2] GA's tracking agent – the JavaScript that collects and send the data to GA's backend – is found in more than 44% of the page loads of users in Germany; 9 out of 20 times when someone in Germany loads a Web page in a browser or follows a link, GA will receive a signal of such an event. GA has a wide reach but at the same time, or rather because of it, they also take active measures to preserve privacy. GA uses no strong or persistent user identifiers. Additionally, GA carries out sanitization of data sent by its tracker, removing elements that could be Personal Identifiable Information (PII). From all trackers that we have analyzed GA is the most privacy friendly one, and yet, privacy of the users is compromised.

GA offers a wide range of analytics to site owners that agree to add GA's tracking script to their sites. Let us describe two use-cases;

1. **Count unique visitors to a page**, i.e. are 10 page loads from 10 different persons, or is it the same person loading the page 10 times? Counting uniques is a basic feature of any analytics framework.

2. **Measure goal completions**. Goals are the basic operations in online marketing to see what was the trigger that led to a particular action which we want to monitor. For instance, did the download of an app happen because the user read about it on a blog post or because the user watched a promotional video?

To be able to satisfy the two use-cases some sort of session of the user's activity has to exist somewhere, how else can GA know whether the user has already visited a page or if it is a new one? Or how can GA track back what a user did after a certain event occurs? Since GA tracking agent does not keep any state, the user's session can only be kept on the collector's back-end (on the server-side). To attribute the signals received to the right user session, a user identifier (`uid`) must exist.

This *uid* will allow GA to link enough signals to build the session required to satisfy a certain use-case, e.g. to calculate goal conversions. The big problem here is that the *uid* can also be used to link all signals from the same user, hence building a session which goes much beyond the scope of the advertised use-cases of GA. The type of *uid* will determine the size of the session. Strong and persistant *uid*s, like the ones used by Facebook, can build sessions spanning to months or even years. The larger the session the more likely it is that one of the records in the session contains some Personal Identifiable Information from which the real identity of the user could be revealed, which means once a record is compromised the whole session is compromised too. On the other a hand short and ephemeral *uid* can only

build a session for a long enough time to satisfy the intended use-case, which may last minutes or perhaps hours. This approach is used by GA and is much safer to the user's privacy.

The type of *uid* only affects the size of the session, which plays into the trade-off between safety vs. data accuracy, but this trade-off will not eliminate side-effects. As a matter of fact, we can claim that **one will always incur undesired side-effects as long as server-side aggregation is needed**.

Strong claims require string evidence, unfortunately for the users, this is easy to obtain. Let us keep focusing on GA, although the same exercise can be repeated with other companies with identical results.

Let us say that I visited a couple of pages: a) my personal homepage at `http://about.me/jmpujol`, for which I logged in. And b) `http://www.depressionforums.org/forums/forum/2-suicide-help-please-read-this`, an extremely sensitive page. In both cases GA tracking script is loaded and the browser's window resolution is sent (`vp=1289x819`). Feel free to repeat the steps yourself by monitoring your `HTTP` requests[1]. The browser's window resolution is not a *uid* on isolation since it changes when you resize your browser window but when combined with the *IP* it can be used to determine that two pages were visited by the same person.[2] Knowing that the two pages were visited by the same person should not be a problem because the user remains anonymous, right? Unfortunately that is not the case. After signing in to `about.me` GA started to receive additional data for each page load (`uid=ffd3...be73501`). This data was not present when not logged-in. Therefore, GA has the ability to learn that the anonymous user is able to login to `http://about.me/jmpujol`, thus breaking the anonymity of the session. Consequently, I can have my real name associated to the rest of the pages in that session, including the page about a very sensitive topic.

It is unlikely that GA is really interested in this particular fact about me, nonetheless, they have the ability to learn it. We would like to stress that this is not a one-off case, there are plenty of other examples. For instance, a) the landing page after successful login of a popular dating site in Spain: `http://www.meetic.com/home/index.php`, and b) the analytics page of your `Twitter` account: `https://analytics.twitter.com/user/solso/home`, which also requires login. The `Twitter` page is leaking to GA my identity on Twitter since the page is only accessible to whoever can login as `solso`. From the *uid* GA can derive that both pages were visited by the same person, therefore, they can know that `solso` on Twitter also has an account to a dating site `Meetic`.

As stated above, as long as server-side aggregation of users' data is used, privacy side-effects are unavoidable. The risk can be reduced, as GA does, but they still exists as we just showed twice. The problem is not lack of care, at least not for GA – we would not be able to extend the same consideration to other trackers. The problem is that the approach of *collect-all-you-can* and then aggregate on the server-side is flawed.

The current approach to deal with side-effects is to establish strict data protection and data retention policies to guarantee that sensitive data does not leak or is not misused. This is not a solution to the problem, but merely a contingency plan.

[1]Open your browser *dev console* and the select *network* tab and filter for *google-analytics*.

[2]We are not certain if GA uses the *IP* and browser's window size to build *uids* or if they use another method. In fact it does not even matter. GA needs to have a *uid* to be able to aggregate the data by user of the server-side in order to satisfy their use-cases. How the *uid* is built only affect the trade-off between data accuracy and safety.

3. AN ALTERNATIVE APPROACH

At Cliqz we faced the similar problem when designing our data collection system. We needed data from our users to build our services: a browser with an integrated search engine, news recommendation, security services such as anti-phishing and anti-tracking and so on. However, we were very troubled by the side-effects that come along with data collection. That is why we created the *Human Web* [1], a novel approach to data collection that relies on client-side aggregations rather than server-side. Since server-side aggregation of users' data is bound to produce side-effects, it is strictly forbidden. Our data collection back-end only receives signals from our users if and only if those signals require no further aggregation. If aggregation at a user level is required, it is carried out in the client itself. Thus no `uid` ever reaches our data collection back-end. Apart from removing explicit *uids* we have a complex set of heuristics to detect potential implicit *uids* in the content of the signals. We also take care of communication-level identifiable parameters by running all signals through a set of anonymization proxies that strip network information which could be used for fingerprinting. We also consider and remove temporal and spatial correlation caused by time of reception and order of arrival of the signals. Thanks to the *Human Web*, we safely collect more than 10M signals per day from our 500K users in Germany. The biggest chunk of signals correspond to pages (6M daily) followed by queries (2M daily)[3]. This paper, however, is not about our *Human Web*, which has been in production for more than a year. The motivation of this paper is to showcase that our approach can be generalized to other use-cases besides Cliqz. Let us take the simple use-case of counting daily unique visitors to a page,

1) With the *server-side aggregation* approach, the backend will receive signals indicating visits to pages with a *uid*, e.g. visiting `https://about.me/jmpujol` twice will send:

```
google-analytics.com/collect?.....dl=https://about.me/
    jmpujol&.....&vp=1289x819.....&_u=AACAAAABI&.....
google-analytics.com/collect?.....dl=https://about.me/
    jmpujol&.....&vp=1289x819.....&_u=AECAAAABI&.....
```

the backend will count the number of unique combinations of page and the tuple *IP* and `vp` (that acts as `uid`) to determine that the page was visited twice and there was one unique visitor.

2) With the *client-side aggregation* approach the client will only send a signal if it is the first visit on that day, and do nothing if already sent. This will be the only data sent:

```
green-tracker.fbt.co/collect?{"ts":"201601291959","type
    ":"page_load,"p": "https://about.me/jmpujol"}
green-tracker.fbt.co/collect?{"ts":"20160129","type":"
    page_visit_by_day", "p":"https://about.me/jmpujol"}
```

[3]This figures do not included telemetry signals, such as response times or crash reports, which are not subjected to privacy considerations.

```
green-tracker.fbt.co/collect?{"ts":"201601292120","type
    ":"page_load,"p": "https://about.me/jmpujol"}
```

Now to calculate the number of unique visitors the back-end only needs to count the signals with type `page_visit_by_day` and ignore the rest (which are used to count page loads). It is the client itself who now guarantees that it will only send the unique visitor signal once and only once per day page. Therefore there is no need to send a *uid* since the desired aggregation has already been done. To be able to do the client-side aggregation the analytics scripts needs to have access to computation and storage. Cliqz meets these requirements since it is a browser, however, the same can be achieved by any other analytics script.

3.1 A Privacy Preserving GA Clone

The current `HTML5` standard allows for computation and storage without violating Cross Scripting or Same Origin Policies. Therefore, it is possible to create a GA-like tracking script that operates only on client-side aggregation, without side-effects. To demonstrate this we have build a prototype of a GA clone that is able to collect data for a wide variety of use-cases: 1) Unique visitors, both at site and page level. 2) Returning customers and retention. 3) Goal Conversion, even across sites. 4) Cross-site correlations, 5) Intra-site click-through patterns, and 6) Page loads and average time spent on page by visitors.

A demo of the GA clone is available at `http://site1.test.cliqz.com/`. Site owners can add the tracking agent to their pages as,

```
<script> window.onload = function() { document.
    getElementById("gt").contentWindow.postMessage(s,"
    http://green-tracker.fbt.co");}</script>
<iframe id="gt" src="http://green-tracker.fbt.co/frame"
    style="display: none;"/>
```

We encourage you to check the code of the *green-tracker* tracking agent [4]. This script – less than 200 lines with comments and helpers removed – is able to satisfy all the aforementioned use-cases without ever sending a *uid*. Inspection of the code will reveal heavy usage of `HTML5 localStorage`, used to keep a consistent state of the user across all the sites that contain the tracking script. A consistent user state is only possible because the script is loaded as an 3rd party `iframe` rather than as the typical 3rd party `script`. Regardless of which site invokes the tracking script its context will always be the origin, i.e. `green-tracker.fbt.co`, therefore, the `localStorage` is consistent across all sites. This would not be the case if the tracking script is loaded using the `script` tag, since the origin would be the site itself, preventing us to aggregate the user's activity. Running the tracking script on an `iframe` is also more secure, since it is a sandboxed environment. Unlike the real GA tracking script, our clone does not have privileges to access the parent window; it is not possible for us to gather information that is not explicitly allowed by the site owner using `postMessage`, adding control and transparency to both site owners and users. The only potential drawback is that `localStorage` can be inspected by whoever has physical access to the users's computer – the same goes for `cookies`. Thus we must be careful of not creating a parallel history. We address this issue by not storing long-term state as plaintext but as truncated hashes. To sum up, the ability to access the `localStorage` of the origin across multiple sites makes the approach of client-side aggregation not only technically possible, but advisable given the multiple advantages in security and privacy it provides.

[4] `view-source:http://green-tracker.fbt.co/frame`

3.2 Example: Goal Conversion Management

In this section we are going to demonstrate how the *green-tracker* agent can measure goal completion without sending a single *uid* to the data collector backend, hence, without putting the user's privacy at risk.

Let us define a goal ***signup*** as completed if a users visits a ***thank-you page*** after seeing either an advertisement on a page from the same site ***Internal Ads***, or an advertisement on a different site ***External Ads***. Additional constraints for the goal completion are that the time between when the ad was clicked until signup must be shorter than 30 minutes, and that the user should not have visited more than 4 other pages in between. Additionaly, we define the goal to be a one-time only, i.e. the goal can only be converted once per user.

Goals in our system are defined in a similar fashion as in other analytics services such as GA or Piwik. The goal is sent to the cient via template. For the demo setup available at `http://site1.test.cliqz.com/` the template of the goal ***signup*** is:

```
{"name": "signup",
    "target": {
           "url": "site4.test.cliqz.com/page-10.
               html", // When a specific
               location loads.
    },
    "referrer": [
        {"url": "site5.test.cliqz.com/page-10.html
             ", "label": "external ad #1"}, //
             Source of traffic
        {"url": "site4.test.cliqz.com/page-9.html
             ", "label": "internal ad #1"}
    ],
    "session_length": 30, // in minutes.
    "pages_per_session": {
                      "max":"4",
        "min":"1"
    },
    "allow_multiple_completions": false // Can be
          true or false.
}
```

Name is the name of the goal; ***target*** is the goal URL; ***referrer*** are the pages from the conversion funnel; ***session_length*** is the maximum time a user has to complete the goal; ***pages_per_session*** is the max/min number of unique pages that can be traversed before achieving the goal; and ***allow_multiple_completions*** specifies if the goal can be completed multiple times by the same user or not.

The process to check for a goal completion is activated when the user loads a page containing the *green-tracker* agent (for the case of the demo that means all pages from all sites).

Let us use code snippets from the *green-tracker* agent to explain the steps taken to validate a goal completion. First of all, we need to keep a very short-term history of pages visited by the user, limited to pages less than 1 hour old and capped to a maximum of 10 (must be values larger than ***session_length*** and ***pages_per_session.max*** respectively)[5].

This how the history object is created and maintained:

[5] We must be careful of not keeping more history than

```
var history = JSON.parse(localStorage.getItem('gt:
    history') || "[]");
history.unshift([url, timestamp]);
history = history.slice(0, 10);
var tooOld = -1;
for(var i=1;i<history.length;i++) {
  if (gtUtils.timeDiffMinutes(timestamp, history[i][1])
      > 60) {
    tooOld = i;
    break;
  }
}
if (tooOld!=-1) history = history.slice(0, tooOld);
localStorage.setItem('gt:history', JSON.stringify(
    history));
```

The aforementioned goal template is saved in the variable ***campaignGoals***, on which we iterate to check if the current URL is one of the target URLs. For clarity we will assume that all goals are kept on a list, which is not a very efficient structure in a real case scenario because we can have thousands of goals,

```
for(var i=0; i<campaignGoals.length; i++) {
  var goal = campaignGoals[i];
  var sessionTime = goal['session_length'];
  var minPagesPerSession = goal['pages_per_session']['
      min'];
  var maxPagesPerSession = goal['pages_per_session']['
      max'];
}
```

we now need to check if this goal allows multiple completions, if not then ensure that this goal has not been already achieved.

```
 if (goal['allow_multiple_conversions'] || (!goal['
     allow_multiple_conversions'] && !cache[gtUtils.
     hashSHA1(goal['name'])]))
```

Let us consider the case that the goal has not been achieved, the next step is to check if the current url is one of the target URLs.

```
if (goal['target']['url']==history[0][0])
```

the goal specifies that the minimum pages per session is 1 and maximum is 4, therefore, we can clip the short-term history and start iterating over it,

```
for(var j=1;j<Math.min(maxPagesPerSession, history.
    length);j++)
```

because we have a temporal limit we only need to proceed further if the pages in the short-term history are within the session time,

```
var prev = history[j];
if (gtUtils.timeDiffMinutes(timestamp, prev[1]) <
    sessionTime)
```

if the url in the short-term history url (currently iterated upon) is one of the referrers in the goal then we conclude the goal has been completed by the user.

```
if (goal['sources'][z]['url']==prev[0])
```

The next step is to create a key composed by the hash of the goal name and stored in `gt:cache_goals` so that the same goal is not reported twice,

```
cache[gtUtils.hashSHA1(goal['name'])] = true;
localStorage.setItem('gt:cache_goals', JSON.stringify(
    cache));
```

The final step is to issue the message to the *green-tracker* backend with the information about the goal completion,

```
dataToSend.push({ts: timestamp, type: 'goal', 'name':
    goal['name'], 'p': lab}
```

The message sent will look like this,

```
{
       "ts": "201604081234",
       "type": "goal",
       "name": "signup",
       "p": "http://site4.test.cliqz.com/page-9.html
            internal ad #1"
}
```

Note that no information about the user has been ever sent. The goal completion has been calculated entirely in the user's browser and the final message contains no data that jeopardize the user's privacy. To count goal completions the backend only needs to count how many messages of type `goal` has received. The backend can segment by time, by goal, by source, etc. This kind of aggregation is allowed, only users' data aggregation is not allowed, or to use a better word, not possible because the users' data never reaches the backend.

We could use the same code snippets to calculate goal completion using conventional server-side aggregation, like GA does. The only difference – albeit a critical one – is that the short-term history of the user would have to be available on the backend, putting the user's privacy at risk. A big risk, specially since it can avoided by using client-side aggregation.

One last thing to consider is that ***campainGoals*** should not be kept as plain text, because that would mean than any user can see which campaigns are being run; this can raise some concerns among companies running those campaigns because they could monitor each other. This problem, which some would call a feature in favor of transparency, can be easily solved. Both 'target' and 'referrers' can be truncated hashes, so that the urls are obfuscated to all but the owner of the goal. Furthermore, the ***labels*** that contain customer specific information can be replaced by ids to be resolved at collection time in the server-side.

4. CONCLUSIONS

To achieve real privacy preservation we must redesign the data collection process to avoid server-side aggregation on user's data, which goes against the standard mode of operation. Fortunately there are alternatives. Cliqz collects data using client-side aggregation only, hence, preserving the privacy of its users. In this talk we would like to showcase that Cliqz's approach can be generalized to other use-cases. To demonstrate it we provide a working prototype of a Google Analytics clone in which data collection does not produce side-effects with regards to privacy.

5. REFERENCES

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

[2] Z. Yu, S. Macbeth, K. Modi, and J. M. Pujol. Tracking the trackers. In *In Proceedings of the 25th ACM International Conference on World Wide Web*, 2016.

strictly necessary because an attacker with physical access to the browser can read from the LocalStorage