Web Browser Privacy: What Do Browsers Say When They Phone Home?

Douglas J. Leith
School of Computer Science & Statistics,
Trinity College Dublin, Ireland
24th Feb 2020*

***Abstract*—We measure the connections to backend servers made by six browsers: Google Chrome, Mozilla Firefox, Apple Safari, Brave Browser, Microsoft Edge and Yandex Browser, during normal web browsing. Our aim is to assess the privacy risks associated with this back-end data exchange. We find that the browsers split into three distinct groups from this privacy perspective. In the first (most private) group lies Brave, in the second Chrome, Firefox and Safari and in the third (least private) group lie Edge and Yandex.**

I. Introduction

While web browsing privacy has been much studied, most of this work has focussed either on (i) measurement of the web tracking/advertising ecosystem, or (ii) methods for detecting and blocking trackers. For example, see [1], [2], [3] and references therein. This line of work has also included consideration of browser private browsing modes, e.g. [4], [5]. However, all of this work typically assumes that the browser itself is a trustworthy platform, and it is this assumption that we interrogate here.

Browsers do not operate in a standalone fashion but rather operate in conjunction with backend infrastucture. For example, most browsers make use of safe browsing services [6] to protect users from phishing and malware sites. Most browsers also contact backend servers to check for updates [7], to faciltate running of field trials (e.g. to test new features before full rollout), to provide telemetry, and so on [8], [9], [10]. Hence, while users are browsing the web Chrome shares data with Google servers, Firefox with Mozilla servers etc as part of normal internal browser operation.

Before proceeding, it is worth noting that most popular browsers are developed by companies that also provide online services accessed via a browser. For example, Google, Apple and Microsoft all provide browsers but also are major suppliers of online services and of course integrate support for these services into their browsers. Here we try to keep these two aspects separate and to focus solely on the backend services accessed during general web browsing.

Our aim is to assess the privacy risks associated with this back-end data exchange during general web browsing. Questions we try to answer include: (i) Does this data allow servers to track the IP address of a browser instance over time (rough location can be deduced from an IP address, so IP address tracking is potentially a surrogate for location tracking) and (ii) Does the browser leak details of the web pages visited.

We study six browsers: Google Chrome, Mozilla Firefox, Apple Safari, Brave Browser, Microsoft Edge and Yandex Browser. Chrome is by far the most popular browser, followed by Safari and Firefox. Between them these browsers are used for the great majority of web access. Brave is a recent privacy-orientated browser, Edge is the new Microsoft browser and Yandex is popular amongst Russian speakers (second only to Chrome). Notable omissions include Internet Explorer, since this is a largely confined to legacy devices, browsers specific to mobile handsets such as the Samsung browser, and the UC browser which is popular in Asia.

We define a family of tests that are easily reproducible and can be applied uniformly across different browsers and collect data on the network connections that browsers generate in response to these tests, including the content of the connections. In these tests we evaluate the data shared: (i) on first startup of a fresh browser install, (ii) on browser close and restart, (iii) on pasting a URL into the top bar, (iv) on typing a URL into the top bar and (v) when a browser is sitting idle. We note that these tests can be automated and used for browser privacy benchmarking that tracks changes in browser behaviour over time as new versions are released. However, analysis of the content of network connections for identifiers probably cannot be easily automated since it is potentially an adversarial situation where statistical learning methods can easily be defeated.

In summary, based on our measurements we find that the browsers split into three distinct groups from this privacy perspective. In the first (most private) group lies Brave, in the second Chrome, Firefox and Safari and in the third (least private) group lie Edge and Yandex.

Used "out of the box" with its default settings Brave is by far the most private of the browsers studied. We did not find any use of identifiers allowing tracking of IP address over time, and no sharing of the details of web pages visited with backend servers.

Chrome, Firefox and Safari all tag requests with identifiers that are linked to the browser instance (i.e. which persist across browser restarts but are reset upon a fresh browser install). All three share details of web pages visited with backend servers.

*Updated 11th March 2020 to include additional information from discussions with Apple and Microsoft.
Updated 19th March 2020 to note that GAPS cookie is no longer set by Google on first startup of Chrome.

This happens via the search autocomplete feature, which sends web addresses to backend servers in realtime as they are typed[1]. Chrome tags these web addresses with a persistent identifier that allows them to be linked together. Safari uses an emphemeral identifier while Firefox sends no identifiers alongside the web addresses. The search autocomplete functionality can be disabled by users, but in all three browsers is silently enabled by default. Chrome sets a persistent cookie on first startup that is transmitted to Google upon browser restart[2] Firefox includes identifiers in its telemetry transmissions to Mozilla that are used to link these over time. Telemetry can be disabled, but again is silently enabled by default. Firefox also maintains an open websocket for push notifications that is linked to a unique identifier and so potentially can also be used for tracking and which cannot be easily disabled. Safari defaults to a choice of start page that prefetches pages from multiple third parties (Facebook, Twitter etc, sites not well known for being privacy friendly) and so potentially allows them to load pages containing identifiers into the browser cache. Start page aside, Safari otherwise made no extraneous network connections and transmitted no persistent identifiers, but allied iCloud processes did make connections containing identifiers. In summary, Chrome, Firefox and Safari can all be configured to be more private but this requires user knowledge (since intrusive settings are silently enabled) and active intervention to adjust settings.

From a privacy perspective Microsoft Edge and Yandex are much more worrisome than the other browsers studied. Both send identifiers that are linked to the device hardware and so persist across fresh browser installs and can also be used to link different apps running on the same device. Edge sends the hardware UUID of the device to Microsoft, a strong and enduring identifier than cannot be easily changed or deleted. Similarly, Yandex transmits a hash of the hardware serial number and MAC address to back end servers. As far as we can tell this behaviour cannot be disabled by users. In addition to the search autocomplete functionality (which can be disabled by users) that shares details of web pages visited, both transmit web page information to servers that appear unrelated to search autocomplete.

The results of this study have prompted discussions, which are ongoing, of browser changes including allowing users to opt-out of search auto-complete on first startup plus a number of browser specific changes. We also note that we consistently found it much easier to engage with open source browser developers (Chrome, Firefox, Brave) since: (i) inspection of the source code allows browser behaviour that is otherwise undocumented (most functionality setting identifiers etc falls into this category) to be understood and validated, and (ii) it is relatively straightforward to make contact with the software developers themselves to engage in discussion. Interaction with Safari developers is confined to a one-way interface (it specifically says no reply will be forthcoming) that allows posting of suggested feature enhancements, although publication of a tech report draft of this paper subsequently did prompt much more helpful contact from Apple and similarly from Microsoft. Interaction with the privacy contacts that Google, Apple, Mozilla etc publicise was wholly ineffective: either they simply did not reply or the reply was a pro forma message directing us to their product support pages.

[1]The default settings are for Chrome to send addresses to Google servers, Firefox to Google servers and Safari to Google and Apple servers. In general this function shares everything a user enters in the top bar, not just URLs. For example, if users accidently type (or cut and paste) passwords or other secrets in the top bar then these will also be shared.

[2]Update 19th March 2020: this bug has been fixed and the GAPS cookie is no longer set.

II. Threat Model: What Do We Mean By Privacy?

It is important to note that transmission of user data to backend servers is not intrinsically a privacy intrusion. For example, it can be useful to share details of the user device model/version and the locale/country of the device (which most browsers do) and this carries few privacy risks if this data is common to many users since the data itself cannot then be easily linked back to a specific user [11], [12]. Similarly, sharing coarse telemetry data such as the average page load time carries few risks.

Issues arise, however, when data can be tied to a specific user. A common way that this can happen is when a browser ties a long randomised string to a single browser instance which then acts as an identifier of that browser instance (since no other browser instances share the same string value). When sent alongside other data this allows all of this data to be linked to the same browser instance. When the same identifier is used across multiple transmissions it allows these transmissions to be tied together across time. Note that transmitted data always includes the IP address of the user device (or more likely of an upstream NAT gateway) which acts as a rough proxy for user location via existing geoIP services. While linking data to a browser instance does not explicitly reveal the user's real-world identity, many studies have shown that location data linked over time can be used to de-anonymise, e.g. see [13], [14] and later studies. This is unsurprising since, for example, knowledge of the work and home locations of a user can be inferred from such location data (based on where the user mostly spends time during the day and evening), and when combined with other data this information can quickly become quite revealing [14]. A pertinent factor here is the frequency with which updates are sent e.g. logging an IP addtress/proxy location once a day has much less potential to be revealing than logging one every few minutes. With these concerns in mind, one of the main questions that we try to answer in the present study is therefore: Does the data that a browser transmits to backend servers potentially allow tracking of the IP address of a browser instance over time.

A second way that issues can arise is when user browsing history is shared with backend servers. Previous studies have shown that it is relatively easy to de-anonymise browsing history, especially when combined with other data (plus recall that transmission of data always involves sharing of the device IP address/proxy location and so this can be readily combined with browsing data), e.g. see [15], [16] and later studies. The second main question we try to answer is therefore: Does the browser leak details of the web pages visited in such a way that they can be tied together to reconstruct the user browsing history (even in a rough way).

We also pay attention to the persistence of identifiers over time. We find that commonly identifiers persist over four time spans: (i) emphemeral identifiers are used to link a handful of transmissions and then reset, (ii) session identifiers are reset on browser restart and so such an identifier only persists during the interval between restarts, (iii) browser instance identifiers are usually created when the browser is first installed and then persist across restarts until the browser is uninstalled and (iv) device identifiers are usually derived from the device hardware details (e.g. the serial number or hardware UUID) and so persist across browser reinstalls. Transmission of device identifiers to backend servers is obviously the most worrisome since it is a strong, enduring identifer of a user device that can be regenerated at will, including by other apps (so allowing linking of data across apps from the same manufacturer) and cannot be easily changed or reset by users. At the other end of the spectrum, emphemeral identifiers are typically of little concern. Session and browser instance identifiers lie somewhere between these two extremes.

We use the time span of the identifiers employed as a simple yet meaningful way to classify browsers, namely we gather browsers using only emphemeral identifiers into one group (Brave), those which use session and browser instance identifiers into a second group (Chrome, Firefox, Safari) and those which use device identifiers into a third group (Edge, Yandex).

It is worth noting that when location and browsing history can be inferred from collected data then even if this inference is not made by the organisation that collects the data it may be made by third parties with whom data is shared. This includes commercial partners (who may correlate this with other data tin their possession), state agencies and disclosure via data breaches.

An important dimension to privacy that we do not consider here is the issue of giving and revoking consent for data use. Our measurements do raise questions in the context of GDPR regarding whether users have really given informed consent prior to data collection, whether opting out is as easy as opting in, and whether the purposes for which consent has been obtained are sufficiently fine-grained (catch-all consent to all uses being inadmissible under GDPR). However we leave this to future work.

III. Measurement Setup

We study six browsers: Chrome (v80.0.3987.87), Firefox (v73.0), Brave (v1.3.115), Safari (v13.0.3), Edge (v80.0.361.48) and Yandex (v20.2.0.1145). Measurements are taken using two Apple Macbooks, one running MacOS Mojave 10.14.6 and one running MacOS Catalina 10.15. Both were located in an EU country when our measurements were collected. We do not expect browser behaviour to change much across desktop operating systems (e.g. we confirmed that we saw similar behaviour on Windows 10 except for additional connections by Edge) but it is worth noting that the mobile handset versions of browsers may well exhibit different behaviour from the laptop/desktop version studied here e.g. Firefox's privacy policy suggests that additional data is collected on mobile devices.

A. Logging Network Connections

To record the timing of network connections and also to log connections we use the open source appFirewall application firewall [17]. Chrome also often tries to use the Google QUIC/UDP protocol [18] to talk with Google servers and we use the firewall to block these, forcing fallback to TCP, since there are currently no tools for decrypting QUIC connections.

B. Viewing Content Of Encrypted Web Connections

Most of the network connections we observe are encrypted. To inspect the content of a connection we use mitmdump [19] as a proxy and adjusted the firewall settings to redirect all web traffic to mitmdump so that the proxying is transparent to the browsers. We add a mitmdump root certificate to the keychain and change the settings so that it was trusted. In brief, when a new web connection starts the mitmdump proxy pretends to be the destination server and presents a certificate for the target web site that has been signed by the trusted mitmdump root certificate. This allows mitmdump to decrypt the traffic. It then creates an onward connection to the actual target web site and acts as an intermediary relaying requests and their replies between the browser and the target web site while logging the traffic.

Note that it is possible for browsers to detect this intermediary in some circumstances. For example, when Safari connects to an Apple domain for backend services then it knows the certificate it sees should be signed by an Apple root cert and could, for example, abort the connection if it observes a non-Apple signature (such as one by mitmdump). However, we did not see evidence of such connection blocking by browsers, perhaps because Enterprise security appliances also use trusted root certificates to inspect traffic and it is not desirable for browsers to fail in Enterprise environments[3]. That said, it is probably worth bearing in mind that browsers may react by changing the contents of their connections when interception is detected, rather than blocking the connection altogether. In our tests we have few means to detect such changes. One is to compare the sequence of servers which the browser connects to (i) without any proxy and (ii) with the proxy in place, and look for differences. We carry out such comparisons and where differences are observed we note them (minor changes were only observed for Firefox);

C. Connection Data: Additional Material

Since the content of connections is not especially human-friendly they are summarised and annotated in the additional material[4]. The raw connection data is also available on request by sending an email to the author (since it contains identifiers, posting the raw data publicly is probably unwise).

[3] We were a little surprised at the absence of cert pinning and the like. Indeed we had expected to need to use tools such as frida to bypass cert checks but in the end this proved unnecessary. We did need to mark the mitmproxy root cert as trusted in the keychain, as without this browser cert checking raised errors, and in part it is this which suggests Enterprise security (with associated installation of trusted root certs) may be a factor here. We also observed use of cert pinning in processes allied to Safari, as noted below.

[4] Available anonymously at https://www.dropbox.com/s/6pao86s99lt4sow/additional_material.pdf

D. Ensuring Fresh Browser Installs

To start a browser in a clean state it is generally not enough to simply remove and reinstall the browser since the old installation leaves files on the disk. We therefore took care to delete these files upon each fresh install. For most browsers it was sufficient to delete the relevant folders in ~/Library/ApplicationSupport and ~/Library/Caches. However, additional steps were needed for Safari since it is more closely integrated with MacOS. Namely, we delete three folders: ~/Library/Safari (to delete the user profile), ~/Library/Containers/com.apple.Safari/Data/Library/Caches (to clear the web cache), ~/Library/Cookies/* (to delete any cookies) and ~/Library/Containers/com.apple.Safari/Data/Library/SavedApplicationState (to reset the start page to the default rather than the last page visited). Also, Firefox was launched with an initial skeleton profile in folder ~/Library/ApplicationSupport/Firefox rather than simply with no folder present. This skeleton profile was created by running Firefox from the command line and quickly interrupting it. A user.js file was then added with the following entries: user_pref(”security.enterprise_roots.enabled”, true); user_pref(”security.OCSP.enabled”, 0). These settings tell Firefox to trust root certificates from the keychain. While Firefox by default has automatic adaptation to allow root certificates this was observed to lead to changes in the sequence of connections made on startup unless this skeleton profile was used.

E. Test Design

We seek to define simple experiments that can be applied uniformly across the set of browsers studied (so allowing direct comparisons), that generate repoducible behaviour and that capture key aspects of general web browsing activity. To this end, for each browser we carry out the following experiments (minor variations necessitated by the UI of specific browsers are flagged when they occur):

1) Start the browser from a fresh install/new user profile. Typically this involves simply clicking the browser app icon to launch it and then recording what happens. Chrome, Edge and Yandex display initial windows before the browser fully launches and in these cases we differentiate between the data collected before clicking past this window and data collected after.
2) Paste a URL into the browser to bar, press enter and record the network activity. The URL is pasted using a single key press to allow behaviour with minimal search autocomplete (a predictive feature that uploads text to a search provider, typically Google, as it is typed so as to display autocomplete predictions to the user) activity to be observed.
3) Close the browser and restart, recording the network activity during both events.
4) Start the browser from a fresh install/new user profile, click past any initial window if necessary, and then leave the browser untouched for around 24 hours (with power save disabled on the user device) and record network activity. This allows us to measure the connections made by the browser when sitting idle.
5) Start the browser from a fresh install/new user profile, click past any initial window if necessary, and then type a URL into the top bar (the same URL previously pasted). Care was taken to try to use a consistent typing speed across experiments. This allows us to see the data transmissions generated by search autocomplete (enabled by default in every browser apart from Brave).

Each test is repeated multiple times to allow evaluation of changes in request identifiers across fresh installs.

Note that since these tests are easily reproducible (and indeed can potentially be automated) they can form the basis for browser privacy benchmarking and tracking changes in browser behaviour over time as new versions are released.

We focus on the default “out of the box” behaviour of browsers. There are several reasons for this. Perhaps the most important is that this is the behaviour experienced by the majority of everyday users and so the behaviour of most interest. A second reason is that this is the preferred configuration of the browser developer, presumably arrived at after careful consideration and weighing of alternatives. A third reason is that we seek to apply the same tests uniformly across browsers to ensure a fair comparison and consistency of results. Tweaking individual browser settings erodes this level playing field. Such tweaking is also something of an open-ended business (where does one stop ?) and so practical considerations also discourage this.

F. Finding Identifiers In Network Connections

Potential identifiers in network connections were extracted by manual inspection[5]. Basically any value present in requests that changes between requests, across restarts and/or across fresh installs is flagged as a potential identifier. Values set by the browser and values set via server responses are distinguished. Since the latter are set by the server changes in the identifier value can still be linked together by the server, whereas this is not possible with browser randomised values. For browser generated values where possible the code generating these values are inspected to determine whether they are randomised or not. We also try to find more information on the nature of observed values from privacy policies and other public documents and, where possible, by contacting the relevant developers.

IV. Evaluating The Privacy Of Popular Back-end Services Used By Browsers

Before considering the browsers individually we first evaluate the data transmissions generated by two of the backend services used by several of the browsers.

[5]We note that unfortunately analysis of content of network connections for identifiers probably cannot be easily automated since it is potentially an adversarial situation where statistical learning methods can easily be defeated.

A. Safe Browsing API

All of the browsers studied make use of a Safe Browsing service that allows browsers to maintain and update a list of web pages associated with phishing and malware. Most browsers make use of the service operated by Google [6] but Yandex also operates a Safe Browsing service [20] and both operators present essentially the same interface to browsers[6]. In view of its importance and widespread use the privacy of the Safe Browsing service has attracted previous attention, see for example [21], [22] and references therein. Much of this focussed on the original Lookup API which involved sending URLs in the clear and so created obvious privacy concerns. To address these concerns in the newer Update API clients maintain a local copy of the threat database that consists of URL hash prefixes. URLs are locally checked against this prefix database and if a match is found a request is made for the set of full length URL hashes that match the hash prefix. Full length hashes received are also cached to reduce repeat network requests. In this way browser URLs are never sent in full to the safe browing service, and some browsers also add further obfuscation by injecting dummy queries. Broadly speaking, the community seems content with the level of privacy this provides with regard to leaking of user browsing history.

However, there is a second potential privacy issue associated with use of this service, namely whether requests can be linked together over time. Since requests carry the client IP address then linking of requests together would allow the rough location of clients to be tracked, with associated risk of deanonymisation. Our measurements indicate that browsers typically contact the Safe Browsing API roughly every 30 mins to request updates. A typical update request sent to safebrowsing.googleapis.com looks as follows:

```
GET https:
//safebrowsing.googleapis.com/v4/threatListUpdates:fetch
Parameters:
   $req: ChwKDGdvb2dsZWNocm9tZRIMOD...
   $ct: application/x-protobuf
   key: AIzaSyBOti4mM-6x9WDnZIjIeyEU2...
```

Note that the dots at the end of the $req and key values are used to indicate that they have been truncated here to save space.

The key value is linked to the browser type e.g. Chrome or Firefox. Each use different key values, but all requests by, for example Chrome browsers, are observed to use the same value. In our measurements the $req value in observed to change between requests. Public documentation for this API makes no mention of a $req parameter, and so these requests are using a private part of the API. However, the difference from the public API seems minor. Inspection of the Chromium source [23][7] indicates that the $req value is just a base64 encoded string that contains the same data as described in the safebrowsing API documentation [6].

The data encoded within the $req string includes a "state" value. This value is sent to the browser by safebrowsing.googleapis.com alongside updates, and is echoed back by the browser when requesting updates. Since this value is dictated by safebrowsing.googleapis.com it can be potentially used to link requests by the same browser instance over time, and so also to link the device IP addresses over time. That does not mean that this is actually done of course, only that it is possible.

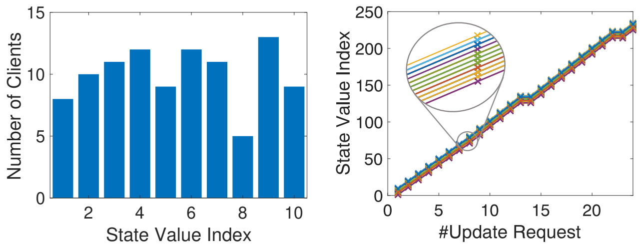
To assist in verifying the privacy of the safe browsing service we note that it would be helpful for operators to make their server software open source. However, this is not currently the case and so to investigate this further we modified a standard Safe Browsing client [24] to (i) use the same key value and client API parameters as used by Chrome (extracted from observed Chrome connections to the Google Safe Browsing service) and (ii) by adding instrumentation to log the state value sent by safebrowsing.googleapis.com in response to update requests. In light of the above discussion our interest is in whether safebrowsing.googleapis.com sends a different state value to each client, which would then act as a unique identifier and facilitate tracking, or whether multiple clients receive the same state value.

A typical state value returned by the safe browsing server is a 27 byte binary value (occasionally longer values are observed). When multiple clients are started in parallel the state values they receive typically differ in the last 5 bytes i.e. they do not receive the same state value. However, closer inspection reveals that each state value is generallly shared by mutliple clients.

For example, Figure 1 shows measurements obtain from 100 clients started at the same time and making update requests roughly every 30 mins (each client adds a few seconds of jitter to requests to avoid creating synchronised load on the server). Since the clients are started at the same time and request updates within a few seconds of each other then we expect that the actual state of the server-side safe browsing list is generally the same for each round of client update requests. However, the clients are not all sent the same value. Instead what happens is that at the first round of requests the 100 clients are assigned to one of about 10 state values. The assignment is not uniform, Figure 1(a) shows the number of clients assigned to each state value, but at least 5 clients are assigned to each. The last 5 bytes of the state value assigned to each client changes at each new update, but clients that initially shared the same state value are assigned the same new value. This behaviour can be seen in Figure 1(b). In this plot we assign an integer index to each unique state value observed, assigning 1 to the first value and then counting upwards. We then plot the state value index of each client vs the update number. Even though there are 100 clients it can be seen from Figure 1(b) that there are only 10 lines, and these lines remain distinct over time (they do not cross). Effectively what seems to be happening is that at startup each client is assigned to one of 10 hopping sequences. Clients assigned to the same sequence then hop between state values in a coordinated manner. Presumably this approach is used to facilitate server load balancing.

[6]Tencent also operate a safe browsing service, see https://www.urlsec.qq.com.

[7]See function GetBase64SerializedUpdateRequestProto() in file components/safe_browsing//core/db/v4_update_protocol_manager.cc



(a) (b)

Fig. 1. State values returned by safebrowsing.googleapis.com over time to 100 clients behind the same gateway. Clients are initialled assigned one out of 10 state values, distributed as shown in (a). Clients assigned the same initial state value are assigned the same state value in subsequent update requests, as shown in (b).

The data shown is for 100 clients running behind the same gateway, so sharing the same external IP address. However, the same behaviour is observed between clients running behind different gateways. In particular, clients with different IP addresses are assigned the same state values, and so we can infer that the state value assigned does not depend on the client IP address.

In summary, at a given point in time safe browsing clients are not all assigned the same state value. However, multiple clients share the same state value, including clients with the same IP address. When there are sufficiently many clients sharing the same IP address (e.g. a campus gateway) then using the state value and IP address to link requests from the same client together therefore seems difficult to achieve reliably. When only one client, or a small number of clients, share an IP address then linking requests is feasible. However, linking requests as the IP address (and so location) changes seems difficult since the same state value is shared by multiple clients with different IP addresses. Use of the Safe Browsing API therefore appears to raise few privacy concerns.

We note that the wording of the Chrome Privacy Whitepaper [8] for safebrowsing.googleapis.com indicates that linking of requests may take place:

> For all Safe Browsing requests and reports, Google logs the transferred data in its raw form and retains this data for up to 30 days. Google collects standard log information for Safe Browsing requests, including an IP address and one or more cookies. After at most 30 days, Safe Browsing deletes the raw logs, storing only calculated data in an anonymized form that does not include your IP addresses or cookies. Additionally, Safe Browsing requests won't be associated with your Google Account. They are, however, *tied to the other Safe Browsing requests made from the same device* [our emphasis].

However, based on our discussions the last sentence likely refers to the transmission of cookies with update requests. Historically, cookies were sent with requests to safebrowsing.googleapis.com [25] but this no longer seems to be the case: we saw no examples of cookies being set by safebrowsing.googleapis.com (and the API documents make no mention on them) and saw no examples of cookies being sent.

B. Chrome Extension (CRX) Update API

Chrome, and other browsers based on Chromium such as Brave, Edge and Yandex, use the Autoupdate API [7] to check for and install updates to browser extensions. Each round of update checking typically generates multiple requests to the update server[8]. An example of one such request is:

```
POST https://update.googleapis.com/service/update2/json
Parameters:
   cup2key=9:2699949029
   cup2hreq=36463a2dd9c$\ldots$39fea17
Headers:
   x-goog-update-appid: mimojjlkmoijpicakmndhoigimigcmbb,
...
   { "request": {
      "@os": "mac",
      "@updater": "chrome",
     "acceptformat": "crx2,crx3",
     "app": [
      {"appid": "mimojjlkmoijpicakmndhoigimigcmbb",
            "brand": "GGRO",
            "enabled": true,
            "ping": { "r": -2},"updatecheck": {},"version":
 "0.0.0.0" },
       <and similar entries>
      "requestid": "{61f7dcb8-474b-44ac-a0e5-b93a621a549b}",
      "sessionid": "{debfbf76-8eaf-4176-82c6-773d46ca8c57}",
      "updaterversion": "80.0.3987.87"}}
```

The appid value identifies the browser extension and the request also includes general system information (O/S etc). The header contains cup2key and cup2hreq values. Observe also the requestid and sessionid values in the request. If any of these values are dictated by the server then they can potentially be used to link requests by the same browser instance together over time, and so also link the device IP addresses over time.

Public documentation for this API is lacking, but inspection of the Chromium source [23] provides some insight. Firstly[9], the cup2key value consists of a version number before the colon and a random value after the colon, a new random value being generated for each request. The cup2hreq is the SHA256 hash of the request body. Secondly, inspection of the Chromium source[10] indicates that in fact the value of sessionid is generated randomly by the browser itself at the start of each round of update checking. The requestid is also generated randomly by the browser[11]. Our measurements are consistent with this: the requestid value is observed to change with each request, the sessionid value remains constant across groups of requests but changes over time. This means that it would be difficult for the server to link requests from the same browser instance over time, and so also difficult to link the device IP addresses of requests over time.

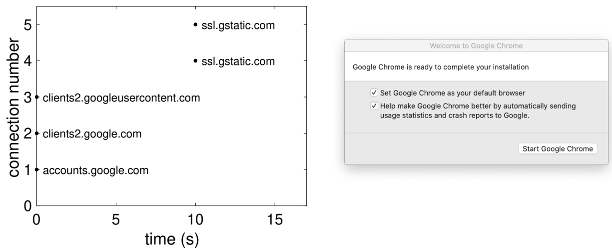
Our measurements indicate that browsers typically check for updates to extensions no more than about every 5 hours.

[8]Chrome sends requests to update.googleapis.com while Brave sends requests to go-updater.brave.com, Edge to edge.microsoft.com and Yandex to api.browser.yandex.com.

[9]See function Ecdsa::SignRequest in file components/client_update_protocol/ecdsa.cc and its call from request_sender.cc

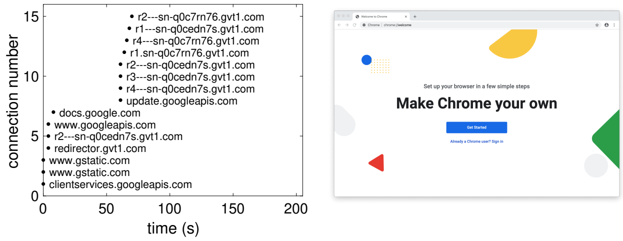
[10]See function UpdateContext::UpdateContext in file components/update_client/update_engine.cc

[11]See function protocol_request::RequestMakeProtocolRequest in file components/update_client/protocol_serializer.cc



(a) (b)

Fig. 2. Chrome connections during first startup, while displays initial popup box as shown. Nothing was clicked.



(a) (b)

Fig. 3. Chrome connections after clicking “start google chrome” in iniital popup shown in Figure 2 (and also clicking to untick the displayed options to make chrome the default browser and allow telemetry). Nothing else was clicked.

V. Data Transmittted On Browser Startup

A. Google Chrome

On first startup Chrome shows an initial popup window, see Figure 2(b). While sitting at this window, and with nothing clicked, the browser makes a number of network connections, see Figure 2(a) for the connections reported by appFirewall.

It is unexpected, and initially concerning, to see connections being made while the popup window asking for permissions is being displayed and has not yet been responded to. However, inspection of the content of these connections indicates that no identifiers or personal information is transmitted to Google, and so while perhaps not best practice they seem to be innocent enough.

After unticking the option to make Chrome the default browser and unticking the option to allow telemetry we then clicked the“start google chrome” button. The start page for Chrome is displayed and another batch of network connections are made, see Figure 3.

Inspection of the content of these connections indicates a device_id value is sent in a call to accounts.google.com, e.g.

```
GET https://accounts.google.com/ServiceLogin
Parameters:
   service: wise
   passive: 1209600
   continue: https://docs.google.com/offline/extension/frame
?ouid
   followup: https://docs.google.com/offline/extension/frame
?ouid
   ltmpl: homepage
Headers:
   x-chrome-id-consistency-request: version=1,client_id
=77185425430.apps.googleusercontent.com, device_id=90c0f8cc-
d49a-4d09-a81c-32b7c0f2aae6, signin_mode=all_accounts,
signout_mode=show_confirmation
```

The device_id value is set by the browser and its value is observed to change across fresh installs, although it is not clear how the value is calculated (it seems to be calculated inside the closed-source part of Chrome). The server response to this request sets a cookie.

The URL http://leith.ie/nothingtosee.html is now pasted (not typed) into the browser top bar. This generates a request to www.google.com/complete/search with the URL details (i.e. http://leith.ie/nothingtosee.html) passed as a parameter and also two identifier-like quantities (psi and sugkey). The sugkey value seems to be the same for all instances of Chrome and also matches the key sent in calls to safebrowsing.googleapis.com, so this is likely an identifer tied to Chrome itself rather than particular instances of it. The psi value behaves differently however and changes between fresh restarts, it therefore can act as an identifier of an instance of Chrome. The actual request to http://leith.ie/nothingtosee.html (a plain test page with no embedded links or content) is then made. This behaviour is reproducible across multiple fresh installs and indicates that user browsing history is by default communicated to Google.

The browser was then closed and reopened. It opens to the Google search page (i.e. it has changed from the Chrome start page shown when the browser was closed) and generates a series of connections, essentially a subset of the connections made on first startup. Amongst these connections are two requests that contain data that appear to be persistent identifiers. One is a request to accounts.google.com/ListAccounts which transmits a cookie that was set during the call to accounts.google.com on initial startup, e.g.

```
 POST https://accounts.google.com/ListAccounts
   cookie: GAPS=1:-qChrMo1Rv\_1fBI0gYpVRkLD\_h89hQ:jJRNg2
Cs370FK-DG
```

This cookie acts as a persistent identifier of the browser instance and since is set by the server changing values can potentially be linked together by the server. The second is a request to www.google.com/async/newtab_ogb which sends an x-client-data header, e.g.

```
 GET https://www.google.com/async/newtab_ogb
   x-client-data:
CIe2yQEIo7bJAQjEtskBCKmdygEIvbDKAQiWtcoBCO21ygEI
jrrKARirpMoBGJq6ygE=
```

According to Google’s privacy documentation[8] the x-client-data header value is used for field trials:

> We want to build features that users want, so a subset of users may get a sneak peek at new functionality being tested before it’s launched to the world at large. A list of field trials that are currently active on your installation of Chrome will be included in all requests sent to Google. This Chrome-Variations header (X-Client-Data) will not contain any personally identifiable information, and will only describe the state of the installation of Chrome itself, including active variations, as well as server-side experiments that may affect the installation. The variations active for a given installation are determined by a seed number which is randomly selected on first run.

(a) (b)

Fig. 4. Firefox connections during first startup, with nothing clicked.

The value of the x-client-data header is observed to change across fresh installs, which is consistent with this documentation. Provided the same x-client-data header value is shared by a sufficiently large, diverse population of users then its impact on privacy is probably minor. However, we are not aware of public information on the size of cohorts sharing the same x-client-data header.

Based on discussions with Google on foot on this work they consider the call to accounts.google.com/ServiceLogin that sets the GAPS cookie to be a bug, likely in the Google Docs extension which is embedded within Chrome, and are currently investigating[12]. We have proposed to Google that an option be added in the initial popup window to allow users to disable search autocomplete before the browser fully launches.

B. Mozilla Firefox

Figure 4(a) shows the connections reported by appFirewall when a fresh install of Firefox is first started and left sitting at the startup window shown in Figure 4(b). The data shown is for startup with mitmproxy running, but an almost identical set of connections is also seen when mitmproxy is disabled (the difference being additional connections to ocsp.digicert.com in the latter case).

During startup Firefox three identifiers are transmitted to Mozilla: (i) impression_id and client_id values are sent to incoming.telemetry.mozilla.org, (ii) a uaid value sent to Firefox by push.services.mozilla.com via a web socket and echoed back in subsequent web socket messages sent to push.services.mozilla.com, e.g.

```
192.168.0.17:62874 <- WebSocket 1 message <-
push.services.mozilla.com:443/
{"messageType":"hello","uaid":"332024
d750734458bc95724268a7b163","status":200,"use_webpush":true
,"broadcasts":{}}
```

These three values change between fresh installs of Firefox but persist across browser restarts. Inspection of the Firefox source code [26] indicates that impression_id and client_id are both randomised values set by the browser[13]. The uaid value is, however, set by the server.

[12]Update 19th March 2020: this bug has been fixed and the GAPS cookie is no longer set.

[13]See function getOrCreateImpressionId() in file browser/components/newtab/lib/TelemetryFeed.jsm for where impression_id is initialised and function _doLoadClientID() in file toolkit/components/telemetry/app/ClientID.jsm for where client_id is initialised.

Once startup was complete, the URL http://leith.ie/nothingtosee.html was pasted into the browser top bar. This generates no extraneous connections.

The browser was then closed and reopened. Closure results in transmission of data to incoming.telemetry.mozilla.org by a helper pingsender process e.g.

```
POST https://incoming.telemetry.mozilla.org/submit/
telemetry/03206176-b1b4-a348-853e-502461c488f7/event/
Firefox/73.0/release/20200207195153?v=4
    <...>
   "reason":"shutdown",<...>
   "sessionId":"cebba8d1-5a4e-d94a-b137-97979fec8c28","
subsessionId":"af1fc2f8-178d-c046-bef9-1d1dea283453",<...>
   "clientId":"0d0214ec-74d8-5640-ae0e-e3dc8952e6aa",
    <...>
```

As can be seen, this data is tagged with the client_id identifier and also contains a sessionId value. The sessionId value is the same across multiple requests. It changes between restarts but is communicated in such a way that new sessionId values can be easily linked back to the old values (the old and new sessionId values are sent together in a telemetry handover message).

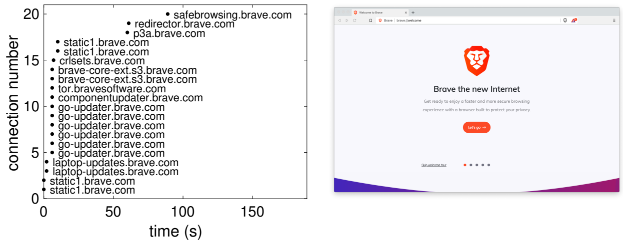
Reopening generates a subset of the connections seen on first start. When the web socket to push.services.mozilla.com is Firefox sends the uaid value assigned to it during first startup to push.services.mozilla.com. Messages are sent to incoming.telemetry.mozilla.org tagged with the persistent impression_id and client_id values.

In summary, there appear to be a four identifiers used in the communication with push.services.mozilla.com and incoming.telemetry.mozilla.org. Namely, (i) client_id and impression_id values used in communication with incoming.telemetry.mozilla.org which are set by the browser and persistent across browser restarts, (ii) a sessionId value used with incoming.telemetry.mozilla.org which changes but values can be linked together since the old and new sessionId values are sent together in a telemetry handover message, (iii) a uaid value that is set by the server push.services.mozilla.com when the web socket is first opened and echoed back in subsequent web socket messages sent to push.services.mozilla.com, this value also persists across browser restarts.

These observations regarding use of identifiers are consistent with Firefox telemetry documentation [9] and it is clear that these are used to link together telemetry requests from the same browser instance. As already noted, it is not the content of these requests which is the concern but rather that they carry the client IP address (and so rough location) as metadata. The approach used allows the client IP addresses/location to be tied together. That does not mean such linking actually takes place, only that the potential exists for it to be done. The Firefox telemetry documentation[14] says that "When Firefox sends data to us, your IP address is temporarily collected as part of our server logs. IP addresses are deleted every 30 days." but is silent on the uses to which the IP data is put.

With regard to the uaid value, Firefox documentation [27] for their push services says uaid is "A globally unique UserAgent

[14]See https://support.mozilla.org/en-US/kb/telemetry-clientid



(a) (b)

Fig. 5. Brave connections during first startup, with nothing clicked.

(a) (b)

Fig. 6. Safari connections during first startup, with nothing clicked.

ID" and "We store a randomized identifier on our server for your browser". We could not find a document stating Mozilla's policy regarding IP logging in their push service.

We have requested clarification from Mozilla of the uses to which the IP metadata logged as part of the telemetry and push services is put, and in particular whether IP addresses are mapped to locations, but have not yet received a response. We have also proposed to Firefox that (i) users are given the option to disable search autocomplete and telemetry on first startup and (ii) the uaid value is not sent over the push.services.mozilla.com websocket until users have registered for one or more push notifications (the web socket itself is apparently also used for distributing notifications related to TLS certificates etc but this does not require a unique identifier).

C. Brave

Figure 5(a) shows the connections reported by appFirewall when a fresh install of Brave is first started and left sitting at the startup window shown in Figure 5(b). During startup no perisistent identifers are transmitted by Brave. Calls to go-updater.brave.com contain a sessionid value, similarly to calls to update.googleapis.com in Chrome, but with Brave this value changes between requests. Coarse telemetry is transmitted by Brave, and is sent without any identifiers attached [28].

Once startup was complete, the URL http://leith.ie/nothingtosee.html was pasted into the browser top bar. This generates no extraneous connections.

The browser was then closed and reopened. No data is transmitted on close. On reopen a subset of the initial startup connections are made but once again no persistent identifiers are transmitted.

In summary, we do not find Brave making any use of identifiers allowing tracking by backend servers of IP address over time, and no sharing of the details of web pages visited with backend servers.

D. Apple Safari

Figure 6(a) shows the connections reported by appFirewall when a fresh install of Safari is first started and left sitting at the startup window shown in Figure 6(b). By default Safari displays a "favorites" page. Resources are fetched for each of the 12 services displayed. This results in numerous connections being made, most of which respond with multiple set-cookie headers. These cookies are scrubbed and not resent in later requests to the prefetched pages. However, we also saw evidence of embedding of identifiers within the prefetched html/javascript, which may then be passed as parameters (rather than as cookies) in requests generated by clicking on the displayed icon.

Once startup was complete, the URL http://leith.ie/nothingtosee.html was pasted into the browser top bar. In addition to connections to leith.ie this action also consistently generated a connection to configuration.apple.com by process com.apple.geod e.g.

```
GET https://configuration.apple.com/configurations/pep/
config/geo/networkDefaults-osx-10.14.4.plist
   User-Agent: com.apple.geod/1364.26.4.19.6 CFNetwork/978.1
 Darwin/18.7.0 (x86_64)
Response is 2.6KB of text/xml
```

This extra connection sent no identifiers.

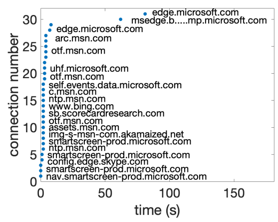
Safari was then closed and reopened. No data is transmitted on close. On reopen Safari itself makes no network connections (the leith.ie page is displayed, but has been cached and so no network connection is generated) but a related process nsurlsessiond consistently connects to gateway.icloud.com on behalf of com.apple.SafariBookmarksSyncAgent e.g.

```
POST https://gateway.icloud.com/ckdatabase/api/client/
subscription/retrieve
  X-CloudKit-ContainerId: com.apple.SafariShared.
WBSCloudBookmarksStore
  X-CloudKit-UserId: _9acd71fb10d466...
  X-CloudKit-BundleId: com.apple.SafariBookmarksSyncAgent
```

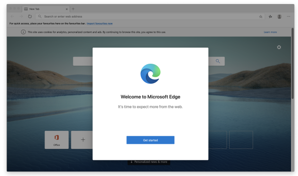
This request transmits an X-CloudKit-UserId header value which appears to be a persistent identifier that remains constant across restarts of Safari. Note that iCloud is not enabled on the device used for testing nor has bookmark syncing been enabled, and never has been, so this connection is spurious. From discussions with Apple it appears this request is unexpected and so may well be a bug.

In summary, Safari defaults to a choice of start page that leaks information to third parties and allows them to cache prefetched content without any user consent. Start page aside, Safari otherwise appears to be quite a quiet browser, making no extraneous network connections itself in these tests and transmitting no persistent identifiers. However, allied processes make connections which appear unnecessary.

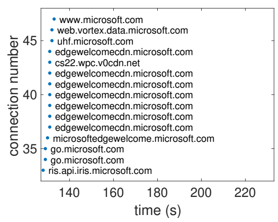
We have proposed to Apple that (i) they change Safari's default start page and (ii) unnecessary network connections by associated processes are avoided.



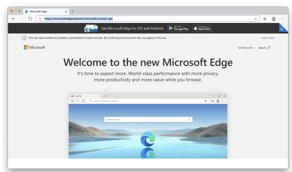
(a) With nothing clicked



(b) Initial screen



(c) After clicking "Get Started" button



(d) Edge welcome page

Fig. 7. Edge connections during first startup.

E. Microsoft Edge

On start up of a fresh install of Edge the browser goes through an opening animation before arriving at the startup window shown in Figure 7(b). Figure 7(a) shows the connections reported by appFirewall during this startup process, with nothing is clicked.

It can be seen that Edge makes connections to a number of Microsoft administered domains. The following observations are worth noting:

1) Fairly early in this process the response to a request to ntp.msn.com includes an "x-msedge-ref" header value which is echoed by Edge in subsequent requests. This value changes on a fresh install of the browser and also across browser restarts, so it seems to be used to tie together requests in a session. Since this vaue is dictated by the server (rather than being randomly generated by the browser) it is possible for the server to also tie sessions together.

2) Much more troubling, later in the startup process Edge sends a POST request to self.events.data.microsoft.com e.g.

```
POST
https://self.events.data.microsoft.com/OneCollector/1.0/
Headers:
   APIKey: 7005b72804a64fa4b...
   SDK-Version: EVT-MacOSX-C++-No-3.2.297.1
Request Body:
   )\x033.0I&Microsoft.WebBrowser.SystemInfo.Configq\x80
\xb4\xaa\xfc\xaa\xe7\xd9\xd7\x11\xa9"o:7005
b72804a64fa4b2138faab88f877b\xd1\x06\x82\x04\xcb\x15
   \x01i\x05Apple\x89\x0eMacBookPro15,2\x00\xcb\x16
   \x01\x00\xcb\x17
   \x01I&u:0B5E1E28-B2E0-5DE9-848D-0368FB...\x00\xcb\x18
   \x01\x89\x08Mac OS X\xa9\x0710.14.6\x00\xcb\x19
   \x01\xa99M:com.microsoft.edgemac_80.0.361.48_x86_64!
Microsoft Edge\xc9\x06\x0b80.0.361.48\x00\xcb\x1f
   \x01I Unmeteredi\x05Wired\x00\xcb
   \x01)\x1bEVT-MacOSX-C++-No-3.2.297.1I$$
eaf6f216-bca7-a0c9-8b40...
   \x01i\x0500:00\x00\xcb%
   \x01\x00\xc9<\x06custom\xcbF
   \x01-
   \x10\x0cAppInfo.ETag\x00\x07Channel0\x00\x91\x08\x00\
x0eConnectionTypei\x07Unknown\x00\x04Etag\x00\
x0fEventInfo.Level0\x00\x91\x04\x00\x0cPayloadClassi\
x0bSYSTEM_INFO\x00\x0b$PayloadGUID
i$2f0dbe5e-a940-4842-8fb3-9b61ed5003ad\x00\
x0ePayloadLogType0\x00\x91
   \x00\x0fappConsentState0\x00\x00\x0bapp_versioni\
x0e80.0.361.48-64\x00 $client_id0\x00\x91\xba\xa1\xa9\
xd4\xef\xc6\xf0\xbc\x04\x00
   installSource0\x00\x00\x0cinstall_date0\x00\x91\x80\
x9c\xcb\xe4\x0b\x00
   pop_sample0\x08\xa8\x00\x00\x00\x00\x00\x00Y@\x00
   session_id0\x00\x91\x02\x00  utc_flags0\x00\x91\x80\
x80\xc0\x80\x80\x80@\x00\x00\x00
```

This request transmits the hardware UUID reported by Apple System Information to Microsoft (highlighted in red). This identifier is unique to the device and never changes, thus it provides a strong, enduring user identifier. This behaviour is consistent with Microsoft documentation [29]. The second block in the request body also contains a number of other identifier-like entries (highlighted in bold since they are embedded within binary content), namely the entries PayloadGUID value and client_id. It is not clear how these values are calculated although they are observed to change across fresh installs.

3) Towards the end of the startup process Edge contacts arc.msn.com. The first request to arc.msn.com transmits a "placement" parameter (which changes across fresh installs) and the response contains a number of identifiers. These returned values are then echoed by Edge in subsequent requests to arc.msn.com and also to ris.api.iris.microsoft.com.

It is not possible to proceed without pressing the "Get Started" button on the popup. Clicking on this button displays a new popup. This new popup has an "x" in the top right corner, and that was clicked to close it. Edge then proceeds to load its welcome page, shown in Figure 7(d). The network connections prompted by these two clicks (the minimal interaction possible to allow progress) are shown in Figure 7(c.

Loading of the Edge welcome page sets a number of cookies. In particular, this includes a cookie for vortex.data.microsoft.com which allows data transmitted to this server to be linked to the same browser instance e.g.

```
GET https://web.vortex.data.microsoft.com/collect/v1/t.js
Parameters:
   <...>
   -impressionGuid: '59d59183-cd90-4787-ab0c-8328d0b20e75'
    <...>
The response sets cookies:
  Set-Cookie: MC1=GUID=da7d27b6947c48b8abd43591e780322d&
HASH=da7d&LV=202002&V=4&LU=1581941544000;Domain=.microsoft.
com;Expires=Tue, 16 Feb 2021 12:12:24 GMT;Path=/;Secure;
SameSite=None
   Set-Cookie: MS0=dc42e1616b0e434e9bef71d2da20f061;Domain=.
microsoft.com;Expires=Mon, 17 Feb 2020 12:42:24 GMT;Path=/;
Secure;SameSite=None
```

The response also includes javascript with the cookie value embedded:

```
   document.cookie="MSFPC=GUID=
da7d27b6947c48b8abd43591e780322d&HASH=da7d&LV=202002&V=4&LU
=1581941544000;expires=Tue,
```

which is used for cross-domain sharing of the cookie (this cookie set by vortex.data.microsoft.com is shared with www.microsoft.com).

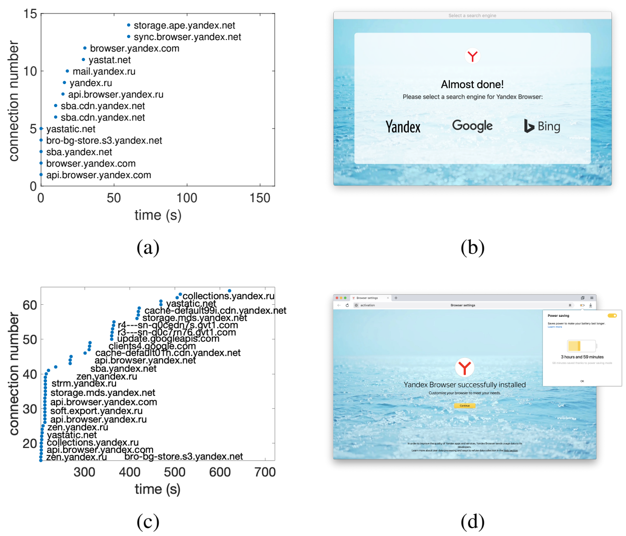


Fig. 8. On first startup Yandex opens an initial popup. When left unclicked no connections were observed. After clicking to deselect the "make default" and "send usage stats" options in the popup, and then clicking "launch" Yandex makes the connections shown in plot (a). Note that Yandex opens a window asking to choose which search engine to use, shown in plot (b). Plot (c) shows the connections made after selecting Yandex as the search engine, plot (d) shows the window displayed after making this click.

At the Edge welcome page the URL http://leith.ie/nothingtosee.html was pasted into the browser top bar. Even this simple action has a number of unwanted consequences:

1) Before navigating to http://leith.ie/nothingtosee.html Edge first transmits the URL to www.bing.com (this is a call to the Bing autocomplete API, and so shares user browsing history with the Bing service of Microsoft). Edge also contacts vortex.data.microsoft.com (which transmits the cookie noted above).

2) After navigating to http://leith.ie/nothingtosee.html Edge then transmits the URL to nav.smartscreen.microsoft.com/, sharing user browsing history with a second Microsoft server.

Edge was then closed and reopened. No data is transmitted on close. On reopen a subset of the connections from the first open are made, including the transmission to self.events.data.microsoft.com of the device hardware UUID for a second time.

F. Yandex Browser

On first startup Yandex opens an initial popup which asks whether the user would like to make Yandex the default browser and whether usage statistics can be shared and which has "Launch" and "Cancel" buttons. While this popup is displayed no network connections were observed.

Figure 8(a) shows the network connections made after clicking to deselect the two options in the popup and then clicking the "Launch" button. The browser makes connections to a number of Yandex administered domains. Early in the startup process the browser sends a cookie on first connecting to yandex.ru. At this point no cookies have been set by server responses so this is presumably a cookie generated by the browser itself. The cookie value is persistent across browser restarts but changes with a fresh browser install, so it acts as a persistent identifier of the browser instance. The response from yandex.ru sets a second cookie and both are sent together in subsequent requests.

At this point the browser displays a screen asking the user to select a search engine, see Figure 8(b). It is not possible to proceed past this point without selecting one. Clicking on the Yandex option generates the network connections shown in Figure 8(c) and brings the browser to the Yandex start page shown in Figure 8(d).

A number of points are worth noting:

1) The browser sends the identifying cookies created on startup to browser.yandex.ru and related domains. As part of this initialisation process a call to browser.yandex.ru/activation/metrika sets a yandexuid cookie for domain yandex.com and a later call to browser.yandex.ru/welcome/ also sets this cookie for domain yandex.ru. This value acts as a persistent identifier of the browser instance (it changes upon a fresh browser install). The two cookies generated at initial startup and this third yandexuid are now sent together with subsequent requests.

2) The browsers sends a client_id and a machine_id value to soft.export.yandex.ru. The client_id value changes across fresh installs. The machine_id value is an SHA1 hash of the MAC address of the device's primary interface and the device serial number i.e. a strong, enduring device identifier. These values are transmitted along with the yandexuid cookie and so they can be tied together.

At the Yandex welcome page the URL http://leith.ie/nothingtosee.html is pasted into the browser top bar:

1) Before navigating to http://leith.ie/nothingtosee.html Yandex first transmits the URL to yandex.ru (this is a call to the Yandex autocomplete API, and so immediately leaks user browsing history to Yandex).

2) After navigating to http://leith.ie/nothingtosee.html Yandex then transmits the text content of http://leith.ie/nothingtosee.html to translate.yandex.net.

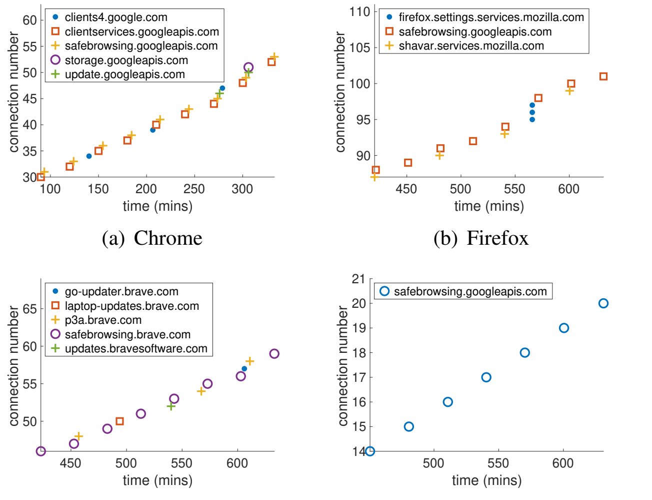
The Yandex browser was then closed and reopened. No data is transmitted on close. On reopen a subset of the connections from the first open are made, transmitting the various cookie values.

VI. Data Transmittted While Browser Is Idle

We now look at the connections made while the browsers are sitting idle for approximately 24 hours. In summary, with the notable exception of Yandex no identifiers are observed to be transmitted in browser backend requests sent while idle.

A. Google Chrome

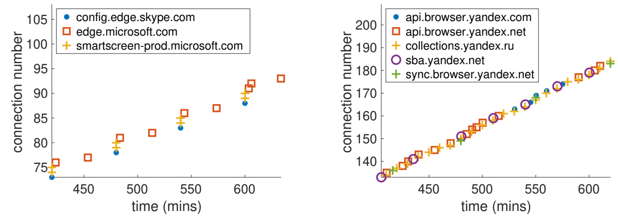
Figure 9(a) shows a typical set of connections made by Chrome reported by appFirewall as the browser sits idle. It can be seen that connects to safebrowsing.googleapis.com (checking for updates to the list of malware URLs) and clientservices.googleapis.com (checking for updates to Chrome field trials)



(a) Chrome (b) Firefox

(c) Brave (d) Safari

Fig. 9. Measurements of browser network connections while browser is sitting idle.



(a) Edge (b) Yandex

Fig. 10. Measurements of browser network connections while browser is sitting idle.

roughly every 30 minutes. About once an hour it connects to clients4.google.com (its not clear what the purpose of this connection is, but it may be related to push notifications). More infrequently (no more than every 5 hours), it connects to update.googleapis.com to check for update to chrome extensions (the connection to storage.googleapis.com shown in Figure 9(a) is on foot of the response from update.googleapis.com directing Chrome to storage.googleapis.com to download an update to an extension) and to www.gstatic.com checking for component updates (Flash, Quicktime, certificate revocation etc).

None of these connections are observed to contain persistent identifers. That said, the connections to clientservices. googleapis.com (checking for updates to field trials) do seem to be made with undue frequency.

B. Mozilla Firefox

Figure 9(b) shows the connections by Firefox reported by appFirewall as the browser sits idle. It can be seen that Firefox connects to safebrowsing.googleapis.com (checking for updates to the list of malware URLs) roughly every 30 minutes and to shavar.services.mozilla.com (checking for updates to Firefox's Tracking Protection blocklists) roughly once per hour. More infrequently, Firefox connects to firefox. settings.mozilla.com (Firefox's remote settiings service [30] for updates to blocklists and for A/B testing and user surveys [10]). These connections occasionally prompt further connections e.g. the response from firefox.settings.mozilla. com can prompt connections to blocklists.settings.services. mozilla.org, services.addons.mozilla.org, aus5.mozilla.org.

None of these connections are observed to contain persistent identifers.

C. Brave

Figure 9(c) shows the connections by Brave as the browser sits idle. It can be see that Brave connects to safebrowsing.brave.com roughly every 30 minutes. Less frequently Brave connects to p3a.brave.com (sending coarse telemetry), updates.bravesoftware.com (checking for updates to Brave), laptop-updates.brave.com and go-updater.brave.com (checking for updates to extensions).

None of these connections are observed to contain persistent identifers.

D. Apple Safari

Figure 9(d) shows the connections by Safari as the browser sits idle. It can be see that Safari connects to safebrowsing. googleapis.com roughly every 30 minutes but otherwise is silent (at least over the 24 hours or so for which the measurements here were collected). Presumably there are no checks for updates, unlike for other browsers, because this checking is performed separately by Apple's AppStore app.

E. Microsoft Edge

Figure 10(a) shows the connections made by Edge as the browser sits idle. It can be seen that Edge connects to edge.microsoft.com roughly every 30 mins and to config. edge.skype.com and smartscreen-prod.microsoft.com roughly every hour. Two types of request are observed to be made to edge.microsoft.com, one making a call to what appears to be a safe browsing API and one checking for updates to browser extensions, using a similar request format to Chrome and Brave. Requests to config.edge.skype.com seem to be checking for updates to Edge itself. Smartcreen is Microsoft's malware/anti-phishing service and so presumably the requests to smartscreen-prod.microsoft.com relate to that.

None of these connections are observed to contain persistent identifers.

F. Yandex Browser

Figure 10(b) shows the connections made by Yandex as the browser sits idle. It can be seen that Yandex makes connections rather more frequently than the other browsers studied here. Connections to collections.yandex.ru and api.browser.yandex. net are made roughly every 10 minutes, and connections to sba.yandex.net roughly every 30 mins and connections to sync.browser.yandex.net roughly once per hour. Intermittently connections are also made to api.browser.yandex.com.

The connections to collections.yandex.net seem to be checking for browser updates and transmit the identifying cookies created during browser startup (see above for details).

The connections to api.browser.yandex.net transmit the client_id and machine_id identifiers and the purpose of these connections is unclear but the URL suggests it is refresh of configuration information.

The connections to sba.yandex.net seem to be calling the Safe Browsing API, or similar, and send no identifiers. The connections to sync.browser.yandex.net seem to be checking for push notifications, and again carry no identfiers.

The connections to api.browser.yandex.com transmit an x-yauuid header and the yp cookie (which can be used to identify the browser instance, see above). Again, the purpose is unclear but the URL suggests refreshing of browser component display details.

VII. Data Transmittted By Search Autocomplete

In this section we look at the network connections made by browsers as the user types in the browser top bar. As before, each browser is launched as a fresh install but now rather than pasting http://leith.ie/nothingtosee.html into the top bar the text leith.ie/nothingtosee.html is typed into it. We try to keep the typing speed consistent across tests.

In summary, Safari has the most aggressive autocomplete behaviour, generating a total of 32 requests to both Google and Apple. However, the requests for Google contain no identifier and those to Apple contain only an ephemeral identifier (which is reset every 15 mins). Chrome is the next most aggressive, generating 19 requests to a Google server and these include an identifier that persists across browser restarts. Firefox is significantly less aggressive, sending no identifiers with requests and terminating requests after the first word, so generating a total of 4 requests to Google. Better still, Brave disables autocomplete by default and sends no requests at all as a user types in the top bar.

In light of these measurements and the obvious privacy concerns they create, we have proposed to the browser developers that on first start users be given the option to disable search autocomplete.

A. Google Chrome

Chrome sends text to www.google.com as it is typed. A request is sent for almost every letter typed, resulting in a total of 19 requests. For example, the response to typing the letter “l” is:


```
["lewis burton","liverpool","love island","linkedin","
littlewoods","lotto","lidl","laura whitmore","lighthouse
cinema","livescore","liverpool fc","lifestyle","little women
","liverpool fixtures","leeds united","love holidays","lewis
 capaldi","lotto.ie","lifestyle sports","ladbrokes"]
```


Each request header includes a psi value which changes across fresh installs but remains constant across browser restarts i.e. it seems to act as a persistent identifier for each browser instance, allowing requests to be tied together.

B. Mozilla Firefox

Firefox sends text to www.google.com as it is typed. A request is sent for almost every letter typed, but these stop after the first word “leith” (i.e. presumably after the dot in the URL is typed) resulting in a total of 4 requests (compared to 19 for Chrome and 32 for Safari). No identifier are included in the requests to www.google.com.

C. Brave

Brave has autocomplete disabled by default and makes no network connections at all as we type in the top bar.

D. Apple Safari

Safari sends typed text both to a Google server clients1.google.com and to an Apple server api-glb-dub.smoot.apple.com. Data is initially sent to both every time a new letter is typed, although transmission to clients1.google.com stops shortly after the first word “leith” is complete. The result is 7 requests to clients1.google.com and 25 requests to api-glb-dub.smoot.apple.com, a total of 32 requests

No identifier are included in the requests to clients1.google.com. However, requests to api-glb-dub.smoot.apple.com include X-Apple-GeoMetadata, X-Apple-UserGuid and X-Apple-GeoSession header values. In our tests the value of X-Apple-GeoMetadata remains unchanged across fresh browser installs in the same location, the X-Apple-UserGuid value changes across fresh installs but remains constant across restarts of Safari. The X-Apple-GeoSession value is also observed to remain constant across browser restarts. From discussions with Apple the X-Apple-UserGuid and X-Apple-GeoSession values are randomised values generated by the user device which are both reset every 15 minutes (by a process external to Safari, hence why they may not change across restarts/fresh installs of Safari that occur within a 15min interval), and this is also consistent with Apple documentation [31]. We have proposed to Apple that this fixed 15min interval be replaced by a randomised interval since otherwise the potential exists to relink identifiers, and so search sessions, by noting which identifiers change within 15 mins of each other (the chance being slim of two devices behind the same gateway choosing exactly the same update time). The X-Apple-GeoMetadata value appears to encode “fuzzed” location [31] but we were unable to verify the nature of the fuzzing used or the (in)accuracy of the resulting location value.

E. Microsoft Edge

Edge sends text to www.bing.com (a Microsoft search service) as it is typed. A request is sent for almost every letter typed, resulting in a total of 25 requests. Each request contains contains a cvid value that is persistent across requests although it is observed to change across browser restarts. Based on discussions with Microsoft, in fact its value changes between search sessions i.e. after the user presses enter in the top bar. Once the typed URL has been navigated to Edge then makes two additional requests: one to web.vortex.data.microsoft.com and one to nav.smartscreen.microsoft.com. The request to nav.smartscreen.microsoft.com includes the URL entered and forms part of Microsoft’s phishing/malware protection service [29], while the request to web.vortex.data.microsoft.com transmits two cookies. From discussions with Microsoft this latter call to web.vortex.data.microsoft.com is made upon navigating away from the welcome page and so does not occur every time a user navigates to a new page.

F. Yandex Browser

Yandex sends text to yandes.ru/suggest-browser as it is typed. A request is sent for every letter typed, resulting in a total of 26 requests. Each request is sent with a cookie containing the multiple identifiers set on Yandex startup. Once the typed URL has been navigated to Yandex then makes two additional requests: one to yandex.ru and one to translate.yandex.ru. The request to yandex.ri sends the domain of the URL entered while the request to translate.yandex.ru sends the text content of the page that has just been visited.

VIII. Conclusions

We study six browsers: Google Chrome, Mozilla Firefox, Apple Safari, Brave Browser, Microsoft Edge and Yandex Browser. For Brave with its default settings we did not find any use of identifiers allowing tracking of IP address over time, and no sharing of the details of web pages visited with backend servers. Chrome, Firefox and Safari all share details of web pages visited with backend servers. For all three this happens via the search autocomplete feature, which sends web addresses to backend servers in realtime as they are typed. In Chrome a persistent identifier is sent alongside these web addresses, allowing them to be linked together. In addition, Firefox includes identifiers in its telemetry transmissions that can potentially be used to link these over time. Telemetry can be disabled, but again is silently enabled by default. Firefox also maintains an open websocket for push notifications that is linked to a unique identifier and so potentially can also be used for tracking and which cannot be easily disabled. Safari defaults to a choice of start page that potentially leaks information to multiple third parties and allows them to preload pages containing identifiers to the browser cache. Safari otherwise made no extraneous network connections and transmitted no persistent identifiers, but allied iCloud processes did make connections containing identifiers.

From a privacy perspective Microsoft Edge and Yandex are qualitatively different from the other browsers studied. Both send persistent identifiers than can be used to link requests (and associated IP address/location) to back end servers. Edge also sends the hardware UUID of the device to Microsoft and Yandex similarly transmits a hashed hardware identifier to back end servers. As far as we can tell this behaviour cannot be disabled by users. In addition to the search autocomplete functionality that shares details of web pages visited, both transmit web page information to servers that appear unrelated to search autocomplete.

References

[1] S. Englehardt and A. Narayanan, "Online tracking: A 1-million-site measurement and analysis," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1388–1401. [Online]. Available: https://doi.org/10.1145/2976749.2978313

[2] W. Meng, B. Lee, X. Xing, and W. Lee, "Trackmeornot: Enabling flexible control on web tracking," in *Proceedings of the 25th International Conference on World Wide Web*, 2016, pp. 99–109. [Online]. Available: https://doi.org/10.1145/2872427.2883034

[3] N. Bielova, "Web tracking technologies and protection mechanisms," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2607–2609. [Online]. Available: https://doi.org/10.1145/3133956.3136067

[4] G. Aggarwal, E. Bursztein, C. Jackson, and D. Boneh, "An analysis of private browsing modes in modern browsers," in *Proceedings of the 19th USENIX Conference on Security*. USENIX Association, 2010.

[5] N.Tsalis, A.Mylonas, A.Nisioti, D.Gritzalis, and V.Katos, "Exploring the protection of private browsing in desktop browsers," *Computers & Security*, 2017.

[6] "Google Safe Browsing API (v4)," 2020. [Online]. Available: https://developers.google.com/safe-browsing/v4

[7] "Chrome App AutoUpdate API," 2020, accessed 21 Feb 2020. [Online]. Available: https://developer.chrome.com/apps/autoupdate

[8] "Chrome Privacy White Paper (January 09, 2020)," 2020. [Online]. Available: https://www.google.com/chrome/privacy/whitepaper.html

[9] "Firefox Telemetry API," 2020, accessed 21 Feb 2020. [Online]. Available: https://firefox-source-docs.mozilla.org/toolkit/components/telemetry/

[10] "Firefox Normandy API," 2020, accessed 21 Feb 2020. [Online]. Available: https://mozilla.github.io/normandy/

[11] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 557–570, 2002.

[12] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "l-diversity: Privacy beyond k-anonymity," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, pp. 3–es, 2007.

[13] G. P. and P. K, "On the Anonymity of Home/Work Location Pairs," in *Pervasive Computing*, 2009.

[14] M. Srivatsa and M. Hicks, "Deanonymizing mobility traces: Using social network as a side-channel," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 628–637.

[15] Z. Weinberg, E. Y. Chen, P. R. Jayaraman, and C. Jackson, "I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks," in *2011 IEEE Symposium on Security and Privacy*. IEEE, 2011, pp. 147–161.

[16] ?ukasz Olejnik, C. Castelluccia, and A. Janc, "Why johnny can't browse in peace: On the uniqueness of web browsing history pattern," in *In Hot topics in Privacy Enhancing Technologies*, 2012.

[17] "appFirewall (v2.02)," 2020, accessed 21 Feb 2020. [Online]. Available: https://github.com/doug-leith/appFirewall

[18] "QUIC, a multiplexed stream transport over UDP," 2020, accessed 21 Feb 2020. [Online]. Available: https://https://www.chromium.org/quic

[19] A. Cortesi, M. Hils, T. Kriechbaumer, and contributors, "mitmproxy: A free and open source interactive HTTPS proxy (v5.01)," 2020. [Online]. Available: https://mitmproxy.org/

[20] "Yandex Safe Browsing API," 2020. [Online]. Available: https://tech.yandex.com/safebrowsing/

[21] T. Gerbet, A. Kumar, and C. Lauradoux, "A Privacy Analysis of Google and Yandex Safe Browsing," in *Proceedings of 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. The publisher of the proceedings, 2016, pp. 347–358.

[22] H. Cui, Y. Zhou, C. Wang, X. Wang, Y. Du, and Q. Wang, "PPSB: An Open and Flexible Platform for Privacy-Preserving Safe Browsing," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[23] "Chromium Source Code," 2020, accessed 21 Feb 2020. [Online]. Available: https://github.com/chromium/chromium

[24] "Reference Implementation for the Usage of Google Safe Browsing APIs (v4)," 2020, accessed 21 Feb 2020. [Online]. Available: https://github.com/google/safebrowsing

[25] "Issue 103243: Cookies no longer sent with safebrowsing," 2011. [Online]. Available: https://bugs.chromium.org/p/chromium/issues/detail?id=103243

[26] "Firefox Source Code (v 73.0)," 2020, accessed 21 Feb 2020. [Online]. Available: https://archive.mozilla.org/pub/firefox/releases/73.0/

[27] "Firefox Push API," 2020, accessed 21 Feb 2020. [Online]. Available: https://mozilla.github.io/application-services/docs/push/welcom.html

[28] "Brave P3A Telemetry API," 2020, accessed 21 Feb 2020. [Online]. Available: https://github.com/brave/brave-browser/wiki/P3A

[29] "Microsoft Edge Privacy Whitepaper," 2020, accessed 6 March 2020. [Online]. Available: https://docs.microsoft.com/en-us/microsoft-edge/privacy-whitepaper

[30] "Firefox Remote Settings API," 2020, accessed 21 Feb 2020. [Online]. Available: https://firefox-source-docs.mozilla.org/main/latest/services/common/services/RemoteSettings.html

[31] "Apple Privacy Features: Suggestions in Search and Safari," 2020, accessed 6 March 2020. [Online]. Available: https://www.apple.com/privacy/features/