This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3065243, IEEE Access

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Digital Object Identifier

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

DOUGLAS J. LEITH[1]
[1]School of Computer Science and Statistics, Trinity College Dublin, Ireland
Corresponding author: Douglas J. Leith (e-mail: doug.leith@tcd.ie).

ABSTRACT We measure the data sent to their back-end servers by five browsers: Google Chrome, Mozilla Firefox, Apple Safari, Brave Browser and Microsoft Edge, during normal web browsing on both desktop and mobile devices. Our aim is to assess the privacy risks associated with this data exchange between a browser and its back-end servers. With regard to shared services, all of the browsers make use of a safe browsing service to mitigate phishing attacks and our measurements indicate that this raises few privacy concerns. Similarly, with regard to the Chrome extension update service accessed by Chromium-base browsers (Chrome, Brave, Edge). Overall, we find that both the desktop and mobile versions of Brave do not use any identifiers allowing tracking of IP address over time, and do not share details of web pages visited with backend servers. In contrast, Chrome, Firefox, Safari and Edge all share details of web pages visited with backend servers. Additionally, Chrome, Firefox and Edge all share long-lived identifiers that can be used to link connections together and so potentially allow tracking over time. In the case of Edge these are device and hardware identifiers that are hard/impossible for users to change. On mobile devices, but not desktop devices, Firefox also shares device identifiers.

INDEX TERMS Privacy, Web Browsers, Internet Measurement, Safe Browsing, Search Autocomplete, Telemetry

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 infrastructure. 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 facilitate 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. To the best of our knowledge the work reported here is the first measurement study of these backend connections made by browsers.

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 data exchange between a browser and its associated Google, Apple, Mozilla etc servers 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 five browsers: Google Chrome, Mozilla Firefox, Apple Safari, Brave Browser, Microsoft Edge. 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. 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 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

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3065243, IEEE Access

connections. 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.

The results of this study have prompted discussions, which are ongoing, with Google, Apple, Mozilla and Microsoft of browser changes to improve privacy. Where changes have now been made we add a footnote to highlight this.

We collect measurements for both the desktop and mobile (Android and iOS) versions of browsers. Our main measurement results are summarised in Table 1.

	First startup	Page navigation	Restart	Typing URL
Brave				
Chrome	I,C	I,U	I,C	I,U
Chrome Android	I,C,P	I,U	I,C	I,U
Chrome iOS	I,C,P,F,T	I,U	I,C	I,U
Firefox	I,T		I,T	U
Firefox Mobile	I,T,D		I,T	U
Safari	P		I	U
Edge	I,H,C,T	C,U	I,H,T	U
Edge Mobile	I,D,C,T,P	C,U	I,D,T	U

TABLE 1. Summary of data shared by browsers with backend servers. Key: I=instance identifier (reset by a fresh browser install), D=device identifier (requires factory reset of device to change), H=hardware identifier (cannot be changed by user), C=cookie, P=prefetch of popular web pages e.g. youtube (the web sites can embed identifiers within the prefetched pages/javascript), F=firebase, T=telemetry, U=url (and so user browsing history).

A. DESKTOP BROWSER VERSIONS

Used "out of the box" with its default settings we found that Brave did not use identifiers allowing tracking of IP address over time, and did not share details of web pages visited with backend servers. In this regard we found Brave to be by far the most private of the browsers studied.

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. 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 ephemeral 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[3]. 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.

[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]On foot of the work reported here Google has modified Chrome so that the GAPS cookie is no longer set.

[3]On foot of the work reported here Mozilla have redesigned their push service to avoid use of a persistent identifier until a user registers for one or more push notifications.

From a privacy perspective desktop version of Microsoft Edge is much more worrisome than the other desktop browsers studied. Edge sends the hardware UUID of the device to Microsoft, a strong and enduring identifier that cannot be easily changed or deleted and can also be used to link different apps running on the same device. In addition to the search autocomplete functionality (which can be disabled by users) that shares details of web pages visited, Edge transmits web page information to servers that appear unrelated to search autocomplete.

B. MOBILE BROWSER VERSIONS

The mobile version of Brave behaves similarly to the desktop version. However, we found that the mobile version of Firefox transmits long-lived identifiers (the Google advertisingId and AndroidId, that persist across browser re-installs) to two third-party web sites, namely app.adjust.com and android.clients.google.com, and also sends browser identifiers to analytics service api.leanplum.com. In this regard the mobile version of Firefox seems significantly less private that the desktop version. The mobile version of Chrome differs from the desktop version in that it prefetches web content from third-party domains (ebay.ie, rte.ie etc), some of which set cookies. In addition, on iOS Chrome makes connections to app-measurement.com and firebaseinstallations.googleapis.com which send browser instance identifiers. The mobile version of Edge also prefetches web content, some of which sets cookies, as well as sending long-lived identifiers to third-parties app.adjust.com and android.clients.google.com. The mobile version of Edge sends a device identifier that persists across browser re-installs.

II. RELATED WORK

The privacy and security of web browsers has been the subject of a substantial literature. This can broadly be classified as concerned with (i) online tracking by web sites, (ii) attacks against the browser itself and (iii) so-called phishing attacks targetted at users.

Online tracking studies originally focussed on cookies and the associated commercial ecosystem, e.g. see [1], [2], [3], [11], but more recently the use of browser fingerprinting has come to the fore since such tracking appears to be much harder to prevent [12], [13], [14], [15]. Related to this, there has been long-standing interest in mitigating tracking via the browser (or upstream gateway) IP address. Perhaps the most prominent technology in this area is Tor, with the literature

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3065243, IEEE Access

mainly focussed on attacks and corresponding defences/mitigations, e.g. see [16], [17], [18].

Attacks against the browser itself include a range of timing-based side channels that can leak information such as user browsing history [19], [20], [21] and more recently memory-based attacks against the sandbox within which web site javascript is executed [22].

Phishing based attacks are typically an online variant of offline fraud, using misdirection and passing off, e.g. see [23]. In response, most browser developers support use of so-called safe-browsing services (mainly Google's service [6], but Microsoft and Yandex also operate safe their own safe browsing services) which maintain a blacklist of fraudulent web sites that is used by the browser to generate warnings when users navigate to a site on the list. Safe-browsing services are widely deployed and as a result their privacy has attracted attention [24], [25]. This work has focussed on the content of the messages exchanged when using a safe-browsing service, primarily with a view to preventing leakage of user browsing history to servers.

In addition, modern browsers all support a so-called private mode. However, the privacy referred to mainly relate to storage of browser history and cookies, namely browsing history and cookies set when in private mode are discarded when private mode is exited. Browser extensions are also typically blocked by default when in private mode and need to be manually enabled. Reflecting the potential for user misunderstanding of the limited nature of the privacy provided by private mode, most of the academic literature has focussed on user studies, e.g. see [26], [5], [4].

To the best of our knowledge there has been no previous systematic work reporting measurements of the content of messages sent between browsers and their associated backend servers.

III. 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 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 [27], [28]. 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 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 [29], [30] 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 [30]. A pertinent factor here is the frequency with which updates are sent e.g. logging an IP address/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 [31], [32] 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) ephemeral 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 identifier 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, ephemeral 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 ephemeral 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).

IV. MEASUREMENT SETUP

We study five browsers: Chrome (v80.0.3987.87), Firefox (v73.0), Brave (v1.3.115), Safari (v13.0.3), Edge (v80.0.361.48). Measurements are taken using an Apple Macbook running MacOS and on a Google Pixel 2 mobile handset running Android 10 and an iPhone SE running iOS 13.5.1. We also collected a smaller set of measurements on Microsoft Windows 10 where we found that the connections made by the browsers were similar to those on MacOS, with the exception of Microsoft Edge which we observed to make

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3065243, IEEE Access

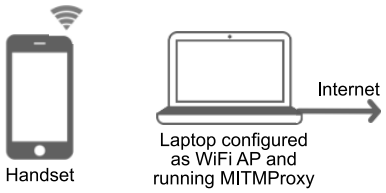


FIGURE 1. Measurement setup used. The user device is configured to access the internet using a WiFi access point hosted on a laptop, use of cellular/mobile data is disabled. The laptop also has a wired internet connection. When an browser on the user device starts a new web connection the laptop pretends to be the destination server so that it can decrypt the traffic. It then creates an onward connection to the actual target server and acts as an intermediary relaying requests and their replies between the browser and the target server while logging the traffic.

additional connections. The devices used are located in Ireland i.e. within a European Union (EU) country. The mobile handsets are connected to the Internet using WiFi. Chrome also often tries to use the Google QUIC/UDP protocol [33] to talk with Google servers and we use a firewall to block these, forcing fallback to TCP, since there are currently no tools for decrypting QUIC connections. Where we observe significant differences between devices we note them below.

A. 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 [34] 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. The setup is illustrated schematically in Figure 1.

Note that it is possible for browsers to detect this intermediary. 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[4].

B. CONNECTION DATA: ADDITIONAL MATERIAL

The content of connections is summarised and annotated in the additional material available anonymously at https://www.dropbox.com/s/gwpzjv6m0mce7ft/browser_privacy_additional_material.pdf.

C. ENSURING FRESH BROWSER INSTALLS

To start a mobile browser in a clean state it is enough to uninstall and then install a new copy of the app since all app files are stored in a directory that is deleted upon uninstall[5]. However, for the desktop versions of browsers we found that old installation files can be left on the disk. We therefore took care to delete these files upon each fresh install.

[4]We did, however, observe use of cert pinning in processes allied to Safari.

[5]On Pixel 2 handsets Chrome is a system app, and similarly Safari on iOS, and so for these we used a factory reset instead.

D. 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 user interface (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 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 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. Note that we observed no identifiers are observed to be transmitted in browser backend requests sent while idle and so to save space we omit further discussion of these results (the connections made are, however, available for inspection in the additional material).

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. Each test is repeated for the desktop version of each browser and for the Android and iOS versions.

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. That said, for due diligence we did confirm that disabling search autocomplete did indeed do that, and similarly disabling telemetry and push notifications.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3065243, IEEE Access

E. FINDING IDENTIFIERS IN NETWORK CONNECTIONS

Potential identifiers in network connections were extracted by manual inspection[6]. 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.

V. 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.

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] and in view of its importance and widespread use the privacy of the Safe Browsing service has attracted previous attention, see for example [24], [25] 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 browsing service, and some browsers also add further obfuscation by injecting dummy queries.

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...
```

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 [35][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.

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 [36] 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 generally shared by multiple clients.

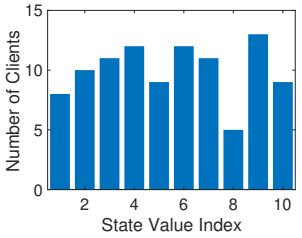
For example, Figure 2 shows measurements obtain from 100 clients started at the same time and making update requests roughly every 30 mins (each client adds a few seconds of jitter to requests to avoid creating synchronised load on the server). Since the clients are started at the same time and request 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 2(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 2(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

[6]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.

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

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3065243, IEEE Access



(a) (b)

FIGURE 2. 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).

can be seen from Figure 2(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.

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.

B. CHROME EXTENSION (CRX) UPDATE API

Chrome, and other browsers based on Chromium such as Brave and Edge , 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...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 [35] 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.

VI. DATA TRANSMITTTED ON BROWSER STARTUP

A. GOOGLE CHROME

On first startup the desktop version Chrome shows an initial popup window. While sitting at this window, and with nothing clicked, the browser makes a number of network connections to various servers at domains (clients2.google.com, ssl.gstatic.com etc) registered to Google, see Figure 3(a). 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.

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

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

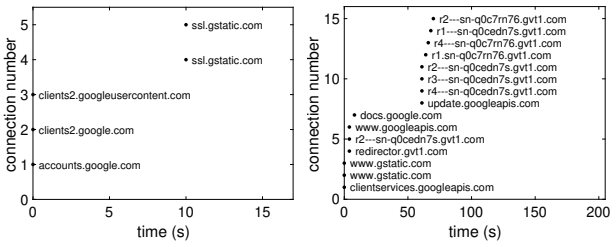
[9]See function Ecdsa::SignRequest in file components/client_update_protocol/ecdsa.cc and its call from request_sender.cc

[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

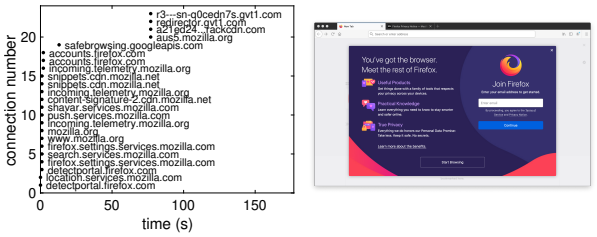
This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3065243, IEEE Access



(a) (b)

FIGURE 3. (a) Chrome connections during first startup with nothing clicked and (b) connections after clicking "start google chrome" in iniital popup.



(a) (b)

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

Chrome is displayed and another batch of network connections are made, see Figure 3(b). Most of the connections in Figure 3(b), e.g to servers at domain gvt1.com, are to the CRX service checking for updates to Chrome extensions but a device_id value is sent in a call to accounts.google.com, e.g.

```
GET https://accounts.google.com/ServiceLogin
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.

On first startup the mobile version of Chrome makes makes connections to m.youtube.com, en.m.wikipedia.org, ir.ebaystatic.com, www.rte.ie, www.smythstoys.com, m.independent.ie to prefetch content, some of which set cookies. In addition, on iOS Chrome makes connections to app-measurement.com and firebaseinstallations.googleapis.com which send browser instance identifiers[12].

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 identifier 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\_1fBI0g...
```

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[13]. 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: CIe2yQEIo7bJAQjEtsk...
```

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

B. MOZILLA FIREFOX

Figure 4(a) shows the connections made when a fresh install of the desktop version of Firefox is first started and left sitting at the startup window shown in Figure 4(b). It can be seen that numerous connection are made to domains (firefox.com, mozilla.com, mozilla.net, mozilla.org etc) registered with Mozilla.

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 [37] indicates that impression_id and client_id

[12]Based on discussions with Google these connections appear to be associated with a bug in Chrome which will fixed as of early 2021.

[13]On foot of the work reported here Google have now modified Chrome to remove setting of this GAPS cookie.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3065243, IEEE Access

are both randomised values set by the browser[14]. The uaid value is, however, set by the server.

In addition, on first startup the mobile version of Firefox generates connections to (i) app.adjust.com which transmit the Google advertising id and an android_uuid, (ii) android.clients.google.com which transmits the device AndroidId, (iii) api.leanplum.com which transmits a deviceId, a uuid and a userId value. The Google advertising id and the AndroidId values persist across fresh browser installs.

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",
    <...>
```

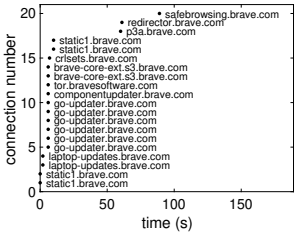
As can be seen, this data is tagged with the client_id identifier and also contains a sessionId value. The sessionId value is the same across multiple requests. It changes between restarts but is communicated in such a way that new sessionId values can be easily linked back to the old values (the old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

[14]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.



(a) (b)

FIGURE 5. (a) Brave connections during first startup, (b) Safari connections during first startup.

that they carry the client IP address (and so rough location) as metadata. In discussions Mozilla say that the IP address data is used "for fraud detection and for disaster recovery purposes".

With regard to the uaid value[15], Firefox documentation [38] for their push services says uaid is "A globally unique UserAgent ID" and "We store a randomized identifier on our server for your browser".

C. BRAVE

Figure 5(a) shows the connections made when a fresh install of Brave is first started. It can be seen that connections are made to a variety of subdomains of brave.com, mostly checking for updates to Chrome components (Brave is based on Chromium, as is Chrome). During startup no persistent identifiers 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 [39].

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 5(b) shows the connections reported by appFirewall when a fresh install of Safari is first started and left sitting at the startup window. By default Safari displays a "favorites" page. It can be seen that connections are made to prefetch content from wikipedia, twitter, yahoo, google, facebook, tripadvisor, linkedin, yelp, weather.com as well to a number of well known ad trackers such as scorecardresearch.com, googlesyndication.com, googletagservices, moadads.com, perfectmarket.com. Most of these connections respond

[15]On foot of the work reported here Mozilla have redesigned their push notification service so that an ephemeral uaid value is now used until a user registers for one orm ore push notifications.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3065243, IEEE Access

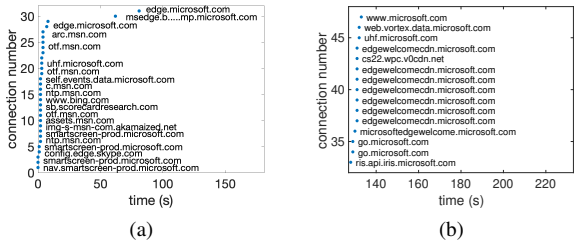


FIGURE 6. Edge connections during first startup (a) with nothing clicked and (b) after clicking "Get Started" button.

with multiple set-cookie headers although 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)
```

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.

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.

E. MICROSOFT EDGE

On start up of a fresh install of Edge the browser goes through an opening animation. Figure 6(a) shows the connections made during this startup process, when nothing is clicked. It can be seen that Edge makes connections to a number of Microsoft administered domains (microsoft.com, msn.com, bing.com) as well as to the ad tracking domain scorecardresearch.com. 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 value 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/
Request Body:
   <...>
   \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...
   <...>
  i$2f0dbe5e-a940-4842-8fb3-9b61ed5003ad\x00\
x0ePayloadLogType0\x00\x91
   \x00\x0fappConsentState0\x00\x00\x0bapp_versioni\
x0e80.0.361.48-64\x00 $client_id00091194600400
   installSource0\x00\x00\x0cinstall_date0\x00\x91\x80
\x9c\xcb\xe4\x0b\x00
   <...>
```

This request transmits the hardware UUID (Universally unique identifier) 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 [40] and has been confimed by Microsoft. 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 a "Get Started" button. Clicking on this button displays a popup which has an "x" in the top right corner, and that was clicked to close it. Edge then proceeds to load its welcome page. The network connections prompted by these two clicks (the minimal interaction possible to allow progress) are shown in Figure 6(b).

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:

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3065243, IEEE Access


```
Set-Cookie: MC1=GUID=da7d27b6947c48b8abd43591e780322d&
HASH=da7d&LV=202002&V=4&LU=1581941544000<...>
Set-Cookie: MS0=dc42e1616b0e434e9bef71d2da20f06<...>
```

The response also includes javascript with the cookie value embedded:

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

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

The mobile version of Edge behaves somewhat differently. The request to self.events.data.microsoft.com that sends a hardware identifier is not made, instead requests often include a log-lived deviceId value which is persistent across fresh browser installs (on iOS this deviceId value is the Apple Advertising Id and on Android it appears to be the android_id). In addition, the mobile version makes connections to: (i) app.adjust.com which transmit the Google advertising id and an android_uuid, and (ii) android.clients.google.com which transmits the device AndroidId. The mobile version of Edge makes connections to www.redditstatic.com, www.wikipedia.org, m.youtube.com, mobile.nytimes.com, www.instagram.com, www.msn.com to prefetch pages, some of which set cookies.

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.

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",<...>,"
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 (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 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 [41]. The X-Apple-GeoMetadata value appears to encode "fuzzed" location [41] but we were unable to verify with Apple the nature of the fuzzing used or the (in)accuracy of the resulting location value.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3065243, IEEE Access

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 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 Smart Screen phishing/malware protection service [40], 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.

VIII. CONCLUSIONS

We present measurements for five browsers: Google Chrome, Mozilla Firefox, Apple Safari, Brave Browser and Microsoft Edge, during normal web browsing on desktop and mobile devices. To the best of our knowledge this is the first public measurement study of the connections made by browsers to their backend servers. All of the browsers make use of a safe browsing service to mitigate phishing attacks and our measurements indicate that this raises few privacy concerns, similarly with regard to the Chrome extension update service accessed by Chromium-base browsers (Chrome, Brave, Edge). For the Brave browser with its default settings we did not find any transmission of identifiers allowing tracking of IP address over time, and no sharing of the details of web pages visited with backend servers. Chrome, Firefox, Safari and Edge all share details of web pages visited with backend servers via the search autocomplete feature. In Chrome a persistent identifier is sent alongside these web addresses, allowing them to be linked together. On iOS Chrome also sends telemetry to backend servers (presumably on Android this function is carried out by Google Play Services rather than Chrome itself), but not on desktop devices. Firefox sends telemetry which includes identifiers that can potentially be used to link these over time. This telemetry can be disabled, but 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. On mobile devices Firefox sends a long-lived device identifier to backend servers, but not on desktop devices. 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. On mobile devices Chrome and Edge similarly prefetch pages (but not on desktop devices). Safari otherwise made no extraneous network connections and transmitted no persistent identifiers, but allied iCloud processes did make connections containing identifiers. On desktop devices Microsoft Edge sends the hardware UUID of the device to Microsoft while on mobile devices it sends long-lived device identifiers. It also collects telemetry tagged with identifiers that can be used to link the telemetry messages over time.

ACKNOWLEDGEMENTS

This work was supported by Science Foundation Ireland grant 16/IA/4610.

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.

[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.

[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.

[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*, 2010.

[5] N. Tsalis, A. Mylonas, A. Nisioti, D. Gritzalis, and V. Katos, "Exploring the protection of private browsing in desktop browsers," *Computers Security*, vol. 67, pp. 181 – 197, 2017.

[6] "Google Safe Browsing API (v4)," 2020, accessed 21 Feb 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, accessed 21 Feb 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] K. Garimella, O. Kostakis, and M. Mathioudakis, "Ad-Blocking: A Study on Performance, Privacy and Counter-Measures," in *Proceedings of the 2017 ACM on Web Science Conference*, 2017, pp. 259–262.

[12] A. Gómez-Boix, P. Laperdrix, and B. Baudry, "Hiding in the Crowd: An Analysis of the Effectiveness of Browser Fingerprinting at Large Scale," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 309–318.

[13] P. Eckersley, "How Unique Is Your Web Browser?" in *Privacy Enhancing Technologies*, M. J. Atallah and N. J. Hopper, Eds., 2010, pp. 1–18.

[14] A. Vastel, P. Laperdrix, W. Rudametkin, and R. Rouvoy, "Fp-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies," in *27th USENIX Security Symposium (USENIX Security 18)*, Aug. 2018, pp. 135–150.

[15] A. Datta, J. Lu, and M. C. Tschantz, "Evaluating Anti-Fingerprinting Privacy Enhancing Technologies," in *The World Wide Web Conference*, ser. WWW '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 351–362.

[16] T. Wang and I. Goldberg, "On Realistically Attacking Tor with Website Fingerprinting," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 4, pp. 21 – 36, 01 Oct. 2016.

[17] S. Feghhi and D. J. Leith, "A Web Traffic Analysis Attack Using Only Timing Information," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 8, pp. 1747–1759, 2016.

[18] ——, "An efficient web traffic defence against timing-analysis attacks," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 2, pp. 525–540, 2019.

[19] T. Van Goethem, W. Joosen, and N. Nikiforakis, "The Clock is Still Ticking: Timing Attacks in the Modern Web," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1382–1393.

[20] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The Spy in the Sandbox: Practical Cache Attacks in JavaScript and Their Implications," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1406–1418.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3065243, IEEE Access

[21] G. Horsman, B. Findlay, J. Edwick, A. Asquith, K. Swannell, D. Fisher, A. Grieves, J. Guthrie, D. Stobbs, and P. McKain, "A forensic examination of web browser privacy-modes," *Forensic Science International: Reports*, vol. 1, 2019.

[22] R. Rogowski, M. Morton, F. Li, F. Monrose, K. Z. Snow, and M. Polychronakis, "Revisiting Browser Security in the Modern Era: New Data-Only Attacks and Defenses," in *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, 2017, pp. 366–381.

[23] A. Oest, Y. Safaei, A. Doup, G. Ahn, B. Wardman, and K. Tyers, "PhishFarm: A Scalable Framework for Measuring the Effectiveness of Evasion Techniques against Browser Phishing Blacklists," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1344–1361.

[24] 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)*, 2016, pp. 347–358.

[25] 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.

[26] H. Habib, J. Colnago, V. Gopalakrishnan, S. Pearman, J. Thomas, A. Acquisti, N. Christin, and L. F. Cranor, "Away From Prying Eyes: Analyzing Usage and Understanding of Private Browsing," in *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, Aug. 2018, pp. 159–175.

[27] 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.

[28] 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, 2007.

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

[30] 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.

[31] 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.

[32] L. 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.

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

[34] A. Cortesi, M. Hils, T. Kriechbaumer, and contributors, "mitmproxy: A free and open source interactive HTTPS proxy (v5.01)," 2020, accessed 21 Feb 2020. [Online]. Available: https://mitmproxy.org/

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

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

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

[38] "Firefox Push API," 2020, accessed 24 Jan 2021. [Online]. Available: https://firefox-source-docs.mozilla.org/browser/components/newtab/docs/v2-system-addon/data_dictionary.html

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

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

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

DOUG LEITH graduated from the University of Glasgow in 1986 and was awarded his PhD, also from the University of Glasgow, in 1989. Prof. Leith moved to the National University of Ireland, Maynooth in 2001 to establish the Hamilton Institute of which he was founding Director from 2001-2014. In 2014, Prof Leith moved to Trinity College Dublin to take up the Chair in Computer Systems in the School of Computer Science and Statistics.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/