2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P) | 978-1-6654-6512-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/EuroSP57164.2023.00058

Been Here Already? Detecting Synchronized Browsers in the Wild

Pantelina Ioannou
University of Cyprus
Nicosia, Cyprus
ioannou.pantelina@ucy.ac.cy

Elias Athanasopoulos
University of Cyprus
Nicosia, Cyprus
athanasopoulos.elias@ucy.ac.cy

***Abstract*—Browsers have become the most popular and used platform for accessing the web. Their wide and exclusive usage as a medium for doing several tasks in the Internet comes with serious security and privacy risks for the users. For example, it has been shown that web sites can employ browser fingerprinting and cross-device tracking techniques to de-anonymizing or profiling a user's browser. On the other hand, browsers become richer in functionality by the years. One very convenient feature, introduced recently and being available to most major web browsers, is synchronizing the browsers on different devices. Browser synchronization allows users to share settings and preferences of their browser running on multiple devices (e.g., on their laptop and smartphone).**

In this paper, we are the first to deliver a framework that can be used by web site operators to detect if different HTTP requests, issued from different browsers, are actually requests performed by the same user through multiple synchronized browsers running on different devices. For detecting this, we reconstruct different sessions based on their requested resources, timestamps and cookies. In addition, we evaluate our methodology by conducting a user study that collects anonymized HTTP requests from several users, and we prove that the detection of synchronized sessions is possible with a success rate higher than 75%. Our results indicate a serious implication to users' privacy that has not been studied before.

***Index Terms*—browser fingerprinting, browser synchronization**

1. Introduction

Browsers are the de facto platforms for accessing the web. Today, users rely on web browsers to access rich services, such as for email, document editing and social interaction. Since web browsers stand as a medium for doing several tasks in the Internet, they have now become a major target for breaching a user's security and privacy.

Users may opt in to access web sites anonymously, without issuing an account for specific web sites. For instance, several news portals or blogs can be viewed without authenticating first. Unfortunately, it has been shown that web sites can employ browser-fingerprinting capabilities and de-anonymize a user, primarily by distinguishing their browser among the many browsers out there [1, 2, 3, 4, 5, 6, 7]. Practically, this means that a user that accesses a web site several times can be *profiled*, even when they are not registered with the web site and, thus, be served with personalized content, such as targeted advertisements. In the worst case, the user's profile may be correlated with the eponymous profile that exists in services that have the user registered. In such cases, anonymous reading of content related to certain political beliefs or lifestyle choices can be entirely de-anonymized and become public.

However, it is crucial to observe that browser fingerprinting is primarily based on distinguishing web browsers within each other and *not* users. For associating a browser to a user, the fingerprinting web site must collude with another web site that has the user registered with. But users today rarely rely on using a *single* browser, since they operate on multiple devices, such as laptops, smartphones, tablets or other smart devices that *do* access the web.

Tracking the user across multiple devices is known as cross-device tracking [8, 9, 10, 11, 12, 13, 14]. To the best of our knowledge, cross-device tracking so far can use several signals (e.g., Bluetooth) but *not* browsers. On the other hand, one very convenient feature available in all major web browsers, today, is *synchronizing* the browsers of different devices. For example, a user can browse the news in the morning using their laptop at the office, then start reading an article, but resume reading after several hours from a different synchronized web browser, let us say from their tablet using their home Internet connection. The extent to which a web site can identify the (anonymous) sessions that belong to *synchronized* browsers of the same user, for eventually cross-tracking the user's browser across different devices, is so far unexplored.

The browser synchronization feature is currently available on Google Chrome, Mozilla Firefox, Microsoft Edge, and Opera. The browsers mentioned cover the 75% of the browser usage worldwide [15]. At the moment, there are no official statistics regarding the usage of browser synchronization. However, it is a feature available in *all* major browser platforms and, most importantly, a feature that has unclear privacy guarantees.

Generalizing typical browser fingerprinting to include synchronized web browsers has significant implications for the user's privacy. First, a web site, which offers content without requiring user authentication, can tell that a user operates specific multiple devices. This may include relatively stationary devices, such as laptops or desktops in a corporate environment, with other relatively transient devices, such as smartphones. Here, the correlated information may include device and network characteristics.

DOI 10.1109/EuroSP57164.2023.00058

For example, a web site can tell that a user employed by a specific company owns a smartphone of a specific brand and lives in a specific area. Second, typical browser fingerprinting can be further augmented with the additional collected information for more aggressive targeted advertisement. As an example, consider a news page that can discover sensitive information about a user, such as religious or political beliefs, based on articles they visit. This information may be used in political campaigns, to manipulate the public opinion.

Our framework could be categorized as a cross-device fingerprinting technique. The core characteristic of cross-device tracking is the detection of the same user in multiple devices. However, the major contribution is that, in contrast to other cross-device tracking works [8, 9, 10, 11, 12, 13, 14], we use the *browser signals* emitted by each synchronized device. For performing the actual tracking, we reconstruct different sessions based on their requested resources and timestamps. We leverage the fact that some web resources cannot be accessed directly by the user, since they are not landing pages. Such resources can be only accessed through several intermediate requests. As an example, consider a web article that is no longer visible in the front page of the main web site, but is archived under a specific theme or category. This article needs a series of steps in order to be accessed again, but if accessed *directly* it may be a signal that the request comes from a synchronized web browser.

Evaluating our methodology is challenging, since we do not have access to existing web sites. For this, we incorporate a novel approach for collecting all web traffic issued by several users that participate in a controlled environment. [1] The traffic is collected anonymously and contains all web accesses from each user that participates in the study. The collected traffic additionally contains synchronization requests to the browser platform used, in our case to Mozilla Firefox. Therefore, we are able to collect the ground truth, i.e., which of the users' browsers are actually synchronized and which are not, and then test the session reconstructions from our methodology. [2]

The total number of requests we collect in this study is 138,962. This amount of data can be compared to the study performed by Laperdix et al. [16] for measuring the effectiveness of tracking with browser fingerprinting. The authors collected 118,934 fingerprints, a sample which is very close to the sample we use in this study. Moreover, the study mentioned above is one of the three large-scale studies performed regarding browser fingerprinting currently available in the literature [2].

The results of our research verify our assumption. From the 138,962 collected data our algorithm detects 76,6% of the user sessions correctly. This means that the 106,420 of sessions are flagged either synced or not synced correctly by the algorithm. The results show that the detection of synced browsers is possible with a high success rate. Moreover, we show that such fingeprinting can be practically realized easily by any web site operator without leveraging active injection of client-side code in the browser's environment. This is a serious implication to users' privacy and, to the best of our knowledge, we are the first to quantify this.

This paper has the following contributions:

- We study the underlying web communication for synchronized web browsers of a popular platform, namely Mozilla Firefox. We can therefore identify synchronization points by detecting specific web requests issued by a web browser towards Mozilla.
- We realize an algorithm that can be used by web operators for correlating different web requests and connecting them to a specific user that controls multiple devices synchronized by a specific web platform.
- We conduct a user study and we assess the effectiveness of our approach. The user study is feasible, since we can construct the ground truth (the synchronized web browsers) by collecting the synchronization points issued by the users' browsers. This information is not available to web operators, but we can use it for evaluating our results. Our results suggest that the detection of synchronized browsers, is possible with a success rate higher than 75%.

2. Background

Device fingerprinting is the identification technique of collecting a list of device features, on several layers of the system, for building a unique identifier for a device. Browser fingerprinting is a similar process, but collecting the information through a web browser, aiming to build a fingerprint of the browser [2]. A browser fingerprint could consist of hardware characteristics, operating system features, and information about the browser and its configuration. The features can either be collected via scripts written in JavaScript or through the passive collection of HTTP headers and network identifiers [2].

Browser fingerprinting acts transparently and, in principle, can produce a state between a server and a web browser similar to the one imposed by cookies. Cookies are the de facto mechanism for facilitating state in HTTP, which is essentially a stateless protocol, by having the server computing a unique token that is attached in all further communication with the visiting client. However, the main difference between cookies and browser fingerprints is that cookies are directly stored inside the browser, whereas browser fingerprinting is a computation performed by the server without the need of cookies. Consequently, browser fingerprinting is increasingly used by companies and organizations, aiming to raise their profits, by presenting clients, even when their cookies are deleted, with personalized content.

2.1. Web browser synchronization

Browser synchronization is a cloud service that is provided by most web browsers, such as Google Chrome, Mozilla Firefox, Microsoft Edge and Opera. When the users synchronize their account, they can share settings and data across multiple devices [17, 18]. Some of the items which can be synchronized are: bookmarks, add-ons and extensions, open tabs, web feeds, credit cards,

1. We have received clearance for performing the study by our national Ethics office, and it covers all the aspects of our study.

2. A fully functional prototype of the plugin's code is available on Bitbucket (https://bitbucket.org/srecgrp/synced_browsers_public/)

and payment methods. The browser synchronization is typically used to maintain a consistent browser setup on multiple devices. The core mechanism behind browser synchronization is that browsers operating in different devices send occasionally probes to the cloud service (e.g., to a Mozilla service) for receiving back all state shared by other browsers operated by the same user. The user needs to operate their browsers while being authenticated with the cloud part. Browser synchronization offers the users with convenience; a resource that is originally opened by one device can be further enabled to other devices. However, browser synchronization can be potentially used by servers to group several different web browsers to a single one, as we explore later in this paper.

3. Detecting Synchronized Browsers

In this section, we describe the approach we follow for reconstructing synchronized web sessions that belong to a single user by a server that receives web traffic from several users. First, we define the threat model and then, we provide an overview of the experimental methodology we follow for validating the possibility of such an attack. Afterwards, we discuss the implementation of the Mozilla extension, which facilitates the collection of needed data for the attack's evaluation.

3.1. Threat model

We assume that the server offers no mandatory authentication mechanism, thus users cannot be identified through their logged-in account shared by multiple devices. Typical web applications that follow this model are news portals and e-commerce sites that can be offered through generic *guest* accounts. However, it is not uncommon for other web applications to support authentication as an optional feature. The server may inject cookies in the web browser, however, these will be different depending on the device the web browser runs. Such a server, that is willing to identify a user's synchronized browsers, acts as an *honest-but-curious* (HBC) adversary. An HBC adversary is defined as a legitimate participant in a communication protocol who does not deviate from the defined protocol but may attempt to learn additional, if possible, information from legitimately received messages [19].

In our case, the adversary is a web site operator that collects all incoming requests and then performs an analysis on them. The goal of the analysis is to potentially learn *more* about the users, and, particularly, which established web sessions can be *grouped* to a single *synchronized* one. More specifically, the web site operator collects retrieved request attributes, such as the URL, timestamp, IP address, user agent, and cookies, and then combines them to reveal which of them are not entirely independent, but are synchronized and, therefore, belong to the same user. This result can be further used for injecting personalized ad content or can be correlated with other web fingerprinting techniques for further de-anonymizing the user. Note that someone can collect attributes from the browser by injecting javascript programs. However, in our study we focus on the attributes that can be collected passively by the web site in the *least intrusive* way. Thus, the attacker in our threat model is passively collecting attributes that are available in the received web traffic.

3.2. Experimental Methodology

The experimental methodology for validating how synchronized browsers can be fingerprinted is depicted in Figure 1. Our first goal is to deploy a technique for collecting several different user sessions, but without revealing any personal or sensitive data. An ideal option for this task is the design and implementation of a browser extension, that collects web traffic generated from users that we know, in advance, if they use synchronized web browsers. We, therefore, implement the browser extension as a Mozilla plug-in, which makes our method compatible with Firefox browsers only. Mozilla Firefox is a popular browser platform, which has been used by 362 million users worldwide (2021) [20], and owns the 11,2% of the browsers usage. It also has the functionality of synchronized browsers, and therefore it is aligned with our objectives. Realizing similar extensions for other web platforms is also possible.

As shown in Figure 1, for collecting all data, users need to install the extension to their Firefox browsers, which may run on different devices. Moreover, users have to provide the plug-in, through a pop-up, with their email address and whether they are synced or not. This data is needed in order to compute the ground truth, i.e., which of the collected sessions are indeed synchronized and they performed by the same user.

Users that install our extension can continue their regular digital routine by visiting any web sites they want. The extension captures all of their web sessions, anonymously, including synchronization requests towards Mozilla, and isolates the attributes we want. Then, the extension constructs a POST request towards a server we control, and adds all the attributes to its body. We can therefore analyse all the collected data for (i) generating the ground truth (how many web sessions are indeed synchronized), (ii) reconstructing all sessions, without the ground truth, using our algorithm, and (iii) comparing the ground truth with our reconstructions.

3.3. Plug-in Design and Implementation

Our browser extension follows the general anatomy of MDN, as reported in [21]. A browser extension must have a manifest.json file, which contains basic metadata for the browser, and declares how the plug-in should behave when added to the user's browser. It contains information such as plug-in description, name, version, the permissions it requires, and pointers to other files. Then, based on what the manifest file contains, we include other files, such as the background page, the browser-action, the page action, and content scripts [21]. For the operation of our extension, we only need a background and a browser-action script. The anatomy of our extension is shown in Figure 2.

The browser extension performs long-term operations, which are included in the background file [21]. The background script of our extension uses the `onBeforeRequest` function [22], to log the URLs. The background script is loaded as soon as the extension

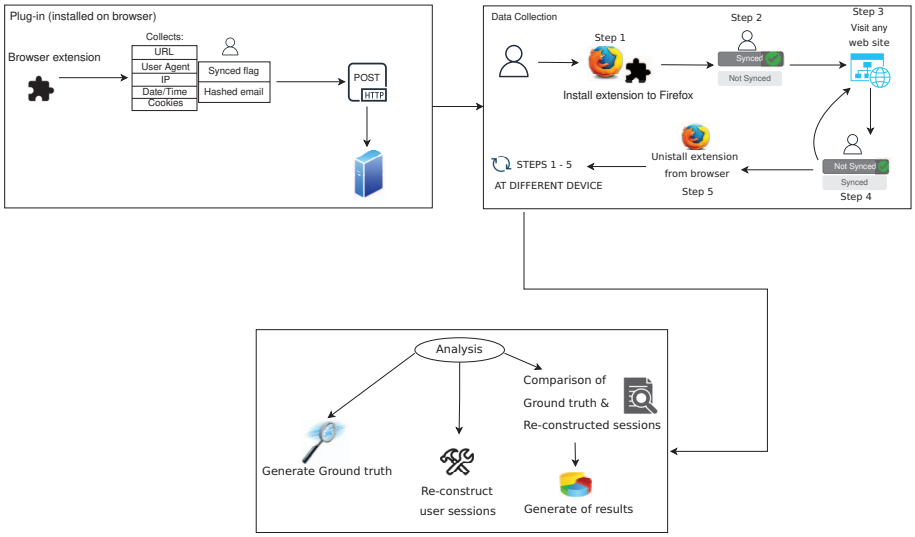

Figure 1: **Data collection flow.** Our experimental methodology is based on three stages. We first design and implement a browser extension which collects four features (URL, user-agent, date/time, IP), the user's email and the synchronization status of the browser. The plug-in sends all collected data to a server. As soon as we have the data, we continue to the analysis phase, which consists of three steps. The generation of the ground truth is done based on the synced flag and the (hashed) email addresses. Additionally, we can analyse the collected requests, based on the URL and the timestamps, in order to reconstruct synchronized sessions. Finally, we compare the ground truth and the reconstructed data.

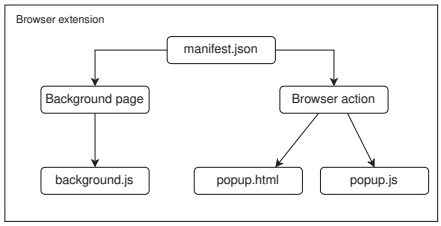

Figure 2: **Extension's anatomy.** The figure above describes the anatomy of the browser extension we developed. The browser plug-in we produced consists of a manifest.json file, a background script, and the files that realize the pop-up window necessary for the email input to the plugin.

is added to the browser, and stays enabled until the user uninstalls the extension. The third component of our extension is the browser action script that contains the popup window. The pop-up appears when the user loads the extension to the browser, and it adds an icon in the right corner of the menu bar.

3.4. Data Collection

We collect the data through the Mozilla browser extension. When the users add the extension to their browser, they have to declare if they are synced/unsynced from a button available in the extension. Then, the extension starts logging the users' requests, and collects the necessary attributes, namely URL, date/time, IP address, User Agent, and Cookies. Users visit any web site they want, continuing their digital routine, for a certain period of time. Then, the users reverse the synced/not synced option and repeat the same steps, as described above. For instance, if the user is logged in their Mozilla account at the first step, they have to log out, choose the "Not synced" button on the extension's pop-up, and start visiting web sites again. Finally, all users are advised to repeat the whole procedure in a different device. The data collection phase is completed when the user uninstalls the plug-in from the browser. An example of a collected request, in raw format, is shown in Figure 3.

Each collected request consists of a header and a body section. The body section is the one we utilize in the analysis phase. As shown in Figure 3, the body section contains the attributes we collect through the plugin: URL, Date, User agent, IP address, and Cookies, which are used in the re-construction algorithm. In the example, we do not include the user_id and some parts of the cookies field, for privacy reasons, to maintain participants' anonymity. The users provide the plugin with their email address, which we cryptographically hash for constructing the user_id. The user_id is subsequently used in the ground truth generation. Furthermore, we log the synced flag which declares whether the user was logged-in to the Mozilla account when performing the specific request.

Although users are not restricted to what web sites can visit, our approach is expected to work with high accuracy with web sites that have dynamic content. For example, news and other similar web sites, where target pages can be visited by following a series of links, which can be later omitted when a synchronized request is re-visiting the final target. We elaborate more on this, in the next section.

```
--SECTION-A--
[17/May/2022:18:42:29 +0200]
Yg569fHFKdNfGpjAeQSbFAAAAAA
10.16.13.119 46366 10.16.20.20 80

--SECTION-B--
POST / HTTP/1.1
Host: *********
User-Agent: Mozilla/5.0
(X11; Linux x86_64; rv:78.0)
Gecko/20100101 Firefox/78.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type:
application/x-www-form-urlencoded
Content-Length: 674
Connection: keep-alive

--SECTION-C--
user_id=*****
& url=http://cnn.com/
& date=17/5/2022, 6:42:29 PM
& userAgent=Mozilla/5.0
(X11; Linux x86_64; rv:78.0)
Gecko/20100101 Firefox/78.0
& ip=194.42.16.139
& synced=false

Cookie Domain: .cnn.com
Cookie Name: countryCode
Cookie Value: CY Persistent: true
Cookie Domain: .cnn.com
Cookie Name: stateCode
Cookie Value: 02 Persistent: true
Cookie Domain: .cnn.com
Cookie Name: geoData
Cookie Value: ****|02|3010|****|EU|20
Persistent: true
--SECTION-E--

--SECTION-Z--
```

Figure 3: **Example of a collected HTTP request.** The figure above shows an example of a collected request in our dataset. The HTTP request contains the header and the body parts. For our methodology we utilize the body part (Section B) of each request, that contains the attributes gathered through the browser extension.

4. Analysis

Once we collect the data we can analyse them for (i) generating the ground truth (how many web sessions are indeed synchronized), (ii) reconstruct all sessions, without the ground truth, using our algorithm, and (iii) compare the ground truth with our reconstructions. The rest of this section describes each of the analysis phases.

4.1. Ground truth and pre-processing

We analyze all the collected HTTP requests, which include also synchronization points, in order to derive the *ground truth*. Synchronization points are identified through the *synced flag* that is provided by the users in the data-collection phase. Users declare specifically when their account is synchronized and when is not. Therefore, the ground-truth set includes all collected HTTP requests, which are also marked as *synced* or *non-synced*. We can use the ground-truth dataset in order to further evaluate the algorithm that attempts to infer synchronized sessions without having access to any of the synchronization points. An example of an anonymized collected HTTP request is shown in Figure 3 and is described in Section 3.4.

A mandatory task in computing the ground truth is removing requests that can introduce types of noise in the final dataset. A prime example of such noise is requests that are not collected through the plug-in and constitute traffic to our web server. These requests can be Internet scans, crawlers or other random traffic received by our web server, which is public. In our processing, we omit all received requests that do not include the `user_id` field, which is attached to every request recorded by our plug-in.

Another type of requests that we exclude from the ground truth is *one-click* requests. One-click requests appear when a user visits a web site only once. Our goal is to infer synced requests that come from, at least, two different devices. Since one-click requests, even when issued by a synced web browser, cannot be used for further inferring if they are paired with requests issued by another device, they are removed from the ground truth.

We filter out all noise and derive the ground-truth table, which consists of a column that indicates whether each request is synchronized or not. If the request is synced, we add a list that includes all the requests that the corresponding session is synced to. If the session is not synced, the field remains `null`. Finally, we derive a new data set, that does not contain the `user_id` and the synced flag for each request. We use this dataset in the reconstruction phase.

4.2. Reconstruction of user sessions

We reconstruct synchronized web sessions by following an intuitive algorithm, which we depict in Figure 4. The diagram reflects how different sessions are correlated using our approach over time. First, a user is visiting a web site at time t_i, and from the landing page of the web site they are likely to generate a series of other requests, in order to access U_{target}. In other words, U_{tagret} is accessed through a *URL path*, U_1 to U_3. For the example shown in Figure 4, the user is reading a specific article from a news web site. The site can be a typical news portal, like CNN, which can be browsed by anyone without authentication. Nevertheless, many other web sites that can be accessed without authentication can be also covered by our algorithm. To signify the importance of the URL path, we use the term *depth*, which in this particular example is equal to three (3).

Now, we can assume that the user saves the article, by utilizing the browser synchronization feature, which

Algorithm 1: Detecting Synchronized Sessions algorithm

```
Result: Reconstructed synced flag for all sessions.
sorted_list = sort_requests_based_on_URL();
keyf = x[URL];
grouped_by_url=[];
for keyf, group in groupby(requests_list, keyf) do
    grouped_by_url. append(list(group));
end
same_timestamps=[];
keyt=x[timestamp];
for keyt, group in groupby(requests_list, keyt) do
    same_timestamps. append(list(group));
end
foreach i ∈ grouped_by_url do
    flag=False
     foreach j ∈ same_timestamps do
        if i.exists(same_timestamps[j]) then
            i.depth=len(same_timestamps[j])
              flag=True
        end
    end
    if flag==False then
        i.depth=1
    end
end
foreach list ∈ grouped_by_url do
    timestamps=[]
     foreach request ∈ list do
        timestamps.append (request.datetime)
    end
    different_timestamps = find_different_timestamps(list,timestamps)
end
similar_cookies=cookies_similarity(different_timestamps)
possible_synced = []
foreach i ∈ different_timestamps do
    foreach j ∈ similar_cookies[i] do
        if i.cookies in similar_cookies then
            possible_synced.append(different_timestamps[i])
        end
    end
end
final_result=[]
foreach request ∈ possible_sync do
    if request.URL.attributes >= 1 then
        final_result.append(request)
    end
end
Function cookies_similarity(different_timestamps):
    individual_cookies = []
    foreach row ∈ different_timestamps do
        foreach request ∈ row do
            individual_cookies.append(different_timestamps[row][request].cookies);
        end
        similar_cookies = []
        foreach i ∈ range(len(individual_cookies) do
            foreach j ∈ range(i + 1, len(individual_cookies) do
                if (individual_cookies[i]==individual_cookies[j]) then
                    similar_cookies.append(individual_cookies[i],individual_cookies[j])
                end
                else if (compare_strings(individual_cookies[i],individual_cookies[j])>=85%) then
                    similar_cookies.append(individual_cookies[i],individual_cookies[j])
                end
            end
        end
    end
    return similar_cookies;
endFunction
```

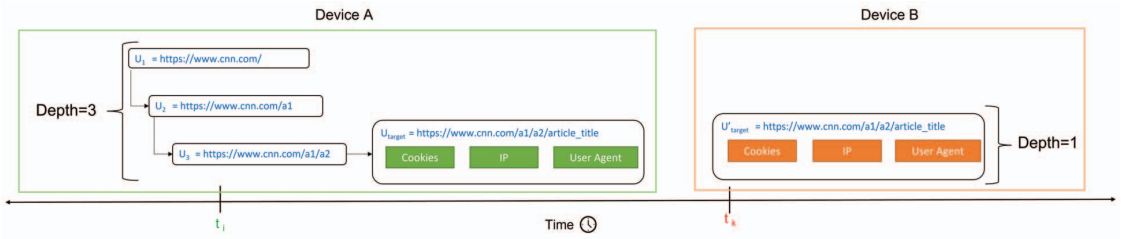

Figure 4: **Session reconstruction.** The timeline of events represents the intuition of our approach. First, a device (A) is visiting a web site at time t_i and is generating a series of actions in order to access U_{target} through a URL path that connects U_1 to U_3. At a later time, t_k, a different device (B) is accessing U'_{target}, but *without* issuing HTTP requests for any of the intermediate steps of the path U_1 to U_3. This action, thus, is signaling an event that the second session, initiated at t_k, is a *synced* session.

allows accessing the article by a different (synchronized) browser running on another device and at any time. Thus, at a later time, t_k, we can assume that the user returns back to the article, but now this comes from a different device. However, this time the user issues an HTTP request towards an article of depth three, without any of the intermediate requests needed. Essentially, the *depth* of the new path is now one (1), and this action is signaling that the session that was performed at time t_k is a *synced* session.

The intuition described above is based on the following observation: when a user visits a web page (e.g., a news portal or an online shopping page) and then they revisit the exact same web page at a different time, in case we have synced browsers, the depth of those two sessions will be different, while the second session's depth will be equal to one. The user re-visits the web page later to continue whatever action did not finish (e.g., continue reading an article or a product purchase order). The two events share the fact that a single page is accessed two times, but the second one is a *direct visit* (depth=1), while the first one is through a URL path (depth > 1).

We realize this methodology in Algorithm 1. The first six lines of the algorithm sort the data, which is then grouped based on the URL attribute. The output is a set of lists that contain all HTTP requests for the exact same URL. The next sixteen lines of code (Lines 7-21) represent how we realize the *depth* characteristic of each request. In lines 7 to 11 we create a list that groups requests that have the exact same timestamp together, in order to detect the URL path of each request. Subsequently, for each request in the `grouped_by_url` list, we search if the request exists in any list from the same timestamps list. If it exists we set the depth of this request equal to the size of the list. Otherwise, if the request is not present in any list of the same timestamps list, we set its depth equal to 1 (Lines 12-21).

We assume that requests with the exact same URL, but with a different timestamp, are likely synced requests. Therefore, we filter out all requests that have the same URL and timestamp. These records reflect multiple requests by a single device. Thus, in lines 22 to 27, we construct a new list that contains all the requests that have the same URL and different timestamps. The `find_different_timestamps` function (Line 26) identifies which requests of the input list have different timestamps. Moreover, before it returns the `different_timestamps` list to the main program, it compares the depth attribute of each request that have been included in the list. If all the requests that have been included to the list have depth equal to 1, then they are removed from the list. If no request have a depth greater than 1, it means that there is no request with a URL path, like it is described in Figure 4.

Apart from the timestamp and the URL, we also use the cookies attribute that is collected through the plug-in. At this point of the algorithm we have identified the requests that are possibly synced based on the characteristics that are described in Figure 4. However, there are cases where visiting a target URL may force the user's browser to perform additional requests towards other URLs. This is common for web pages that include media or ads from other web sites. These subsequent URLs are not user generated, but *machine-generated*. It is difficult to ignore such URLs, since they appear in our logs with arbitrary values of depths and, thus, they are very likely to introduce false positives. To filter out these cases, we leverage the available information of cookies to distinguish machine-generated from user-generated URLs. Cookies are set by the web site when a user visits the target URL. For the machine-generated requests, the URL attribute is the same, but some of the cookies are *not*. This happens because the cookies are generated from web sites hosted in different domains, and subsequently the cookie values differ. For example, the same ad, which can be included when the user is visiting A.com and B.com, has the exact same URL but very different cookies as they are set by A.com and B.com, respectively.

An example of two requests with the characteristics described above is shown in Figure 5. We can observe that the URL of those requests most probably belongs to an ad, generated by the target URL. The cookies field of these two requests does not contain similar information. We can observe that most of the cookies of the first requests have been generated by cnn.com, while most of the cookies of the second requests have been generated from bbc.com and jennikanye.com. Thus, we can conclude that those requests are *machine-generated*, and belong to an ad or media, that are incorporated to the target URL. We remove this kind of requests after the execution of the cookies similarity function. The similarity function, that we use to decide whether the information included in cookies is

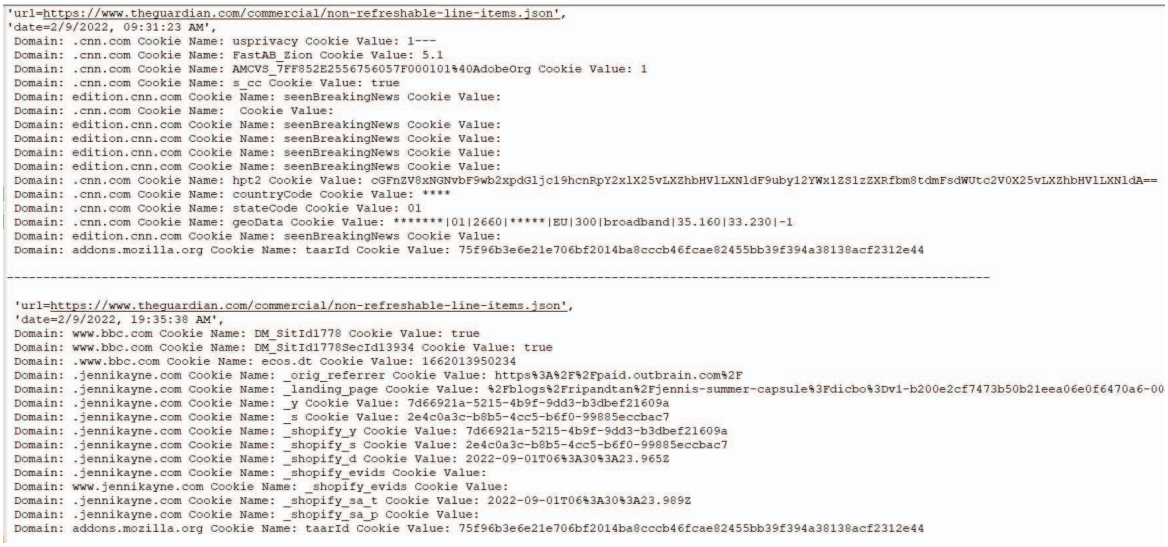

Figure 5: **Cookies similarity example.** The figure above shows an example of the cookies field of two different web sessions, that have the exact same URL and different timestamps. The cookies values differ significantly because the depicted HTTP requests are machine-generated and not user-generated requests, issued by ads or media that are incorporated to the requested web page.

similar, is defined in lines 42 to 60 of Algorithm 1.

First, we separate the cookies for each row of requests that have the same URL and different timestamps, and store it to a list. Subsequently, we use this list to compare the cookies with each other and decide whether they are similar or not. In our case we consider two cookies to be similar when either of the following scenarios applies: (i) the two cookies are exactly the same, or (ii) the two cookies are nearly the same, with a similarity equal or greater than 85%. To achieve that we compare the two strings and search if Cookie Name and Cookie Value fields are the same. If they are the same we increase a counter that indicates that two similar cookies have been found, and we add the similar cookies in a separate list. We repeat the steps for every pair of cookies in the list. When the execution of the similarity function for cookies is completed and it returns to the main program the list that contains the similar cookies, we compare the list that contains the requests with the different timestamps with the `similar_cookies` list (lines 29 to 36). If the cookie for a request is included in the similar cookies list, we add the request to a list with the possible synced requests.

The last part of the algorithm (lines 37 to 42) is to filter out the requests with a URL that does not contain a path. For instance, we keep in our corpus all the requests that indicate a specific web page in www.cnn.com, such as www.cnn.com/news-page115/, but not the main page, www.cnn.com, since it does not contribute to the depth value. The new list that we create, without those requests, contains the possible synced requests of the dataset. This step completes our algorithm, and we consequently create a table with all reconstructed sessions. The rest of the sessions that are not included in the list are considered not synced.

4.3. Comparison

The last step of the analysis computes the percentages of the correct and wrong guesses of our algorithm. For this, we compare the synced field for each request from the ground truth and the reconstruction tables, respectively. An example of how the synced field is stored in the two tables is shown in Figure 8 of Appendix B. The synced field in both tables can either contain a list with the other requests that each HTTP request is synced, or a `null` value, which reflects that the HTTP request is not synced. In case the request is not synced and the algorithm marks it as synced, or vice-versa, then we have a false positive or a false negative, respectively. If the algorithm detects correctly that the request is not synced, meaning that both values of the synced fields are `null`, then we have a true negative. The last case is when the values of both the ground truth and the reconstruction table are not `null`, thus the algorithm detects that the request is synced. In that case, we compare the two lists and if they are identical, we have a true positive. Otherwise, we have a false positive because although the algorithm detects that the request is synced, it fails to compute the correct requests that are synced, which belong to other users.

5. User Study

In this section we describe the procedure of the user study we conduct for evaluating our methodology. The study took place between April and May of 2022; we received a clearance for performing the study from our National Bioethics Committee, in advance. At the end of the study, 30 users participated and we gathered 138,962 HTTP requests in total. The total number of requests we have collected is comparable to one of the three large-scale studies performed on the effectiveness of tracking

with browser fingerprinting [2]. Laperdix et al. perform an analysis of 118,934 fingerprints [16], a sample which is very close to the sample we use in this study. Thus, regardless the number of participants, the volume of our data is comparable to other large-scale studies related to browser fingerprinting.

The users who participate in the experiment have a basic technological background. Furthermore, we do not take into account any demographic information about the users, such as gender, nationality, and age, so the selection of the users is random. By random we mean that we did not choose the users based on any criteria, and the participants of this study are volunteers who accepted our invitation to take part in our study. Initially, each participant needs to go through the aim of the study, and the type of data we collect. We continue the procedure only if the user agrees. Then, each participant needs to install the browser extension we have developed in their browser and follow the instructions we provide to them. The users are not instructed to visit specific web sites, but they are instead free to browse the web. To finish the study they must complete all the steps, as described in Figure 1.

The evaluation of our methodology is a closed-world study, in a controlled environment with a specific number of users. Performing a similar study in an open-world experiment can be challenging. Specifically, users may hesitate to donate freely their web traffic, even if this collection happens anonymously. However, even with a closed-world study, the volume of data collected, as we have already argued, is enough for making the study representative.

6. Evaluation

In this section we evaluate the results of the user study, describe the findings of our work and discuss their importance for user privacy.

6.1. Overview

We compare the ground-truth data with the reconstructed values as they are generated from our algorithm. When comparing the sessions, four scenarios may occur: (i) the ground-truth session is synced and our algorithm detects it as synced, (ii) the ground truth session is not synced and our algorithm does not detect it as synced, (iii) the ground truth session is not synced and our algorithm detects it as synced, and (iv) the ground truth session is synced and our algorithm does not detect it as synced. These scenarios represent the number of true positives, true negatives, false positives, and false negatives, respectively, which we further discuss in Section 4.3.

In the comparison phase we keep four counters, one for each scenario, and we increase the corresponding counter for each session. For instance, if the request is synced in the ground truth and is also synced in the reconstruction table we increase the true positives value. Once the comparison phase is completed, we also compute the total true and false guesses of the study.

The comparison of sessions happens over a balanced dataset in terms of synced and non-synced requests. From the 138,962 collected requests, 64,396 are non-synced and the rest 74,567 are synced. Moreover, we collect five attributes through the browser extension for each session, namely URL, timestamp, user agent, IP address and cookies; we can further compute the value of *depth* for each URL access, as discussed in Section 4.2.

6.2. Performance

Initially, we execute the analysis algorithm as it is discussed in Section 4. Essentially, we use the URL, the timestamp and the computed depth for classifying each session as synced or not synced. The results are represented in Table 1 below.

The total percentage of true guesses is 64%. These results are from 84,070 collected requests from 13 unique users. From the 84,070 requests, 53,553 of them are flagged accurately. More specifically, 41,328 sessions are flagged as synced, and 12,225 sessions as not synced, correctly. The false positives and false negatives reach the 25,059 and 5,458 sessions, respectively. It is evident that the high percentage of false guesses is mostly due to false positives, i.e., sessions that we flag as synced, but they are not. A large fraction of false positives is due to machine-generated requests, as we have discussed in Section 4.2. To reduce false positives, we further leverage the available information of cookies to distinguish machine-generated from user-generated URLs. In the next part, we apply the algorithm using three attributes, namely the URL and timestamp, but, also, the cookies.

6.3. Performance with three attributes

We now consider cookies as a contributing attribute in order to filter out machine-generated from user-generated requests. The results are presented in Table 2. Specifically, from the 138,962 requests we collect in total, the algorithm detects 106,420 of them accurately. The algorithm flags 57,661 sessions as synced, and 48,759 sessions as not synced, correctly. The percentage of true guesses, thus, is $\sim$77%, while the other $\sim$23% corresponds to false guesses. For all false guesses, 15,636 sessions are flagged as synced, but they are not (false positives), while 16,906 synced sessions are not detected (false negatives). Consequently, we have 35,542 false guesses out of 138,962 sessions.

6.4. False guesses

We can observe that there is a significant reduction in false positives when we include cookies in our analysis as described in Section 6.2. From 30% of the total false guesses, we now have only 11.2% of the total guesses. On the other hand, false negatives are increased by 3%, and from 7% of the total guesses, they are now 11%. In order to further attribute the root of false positives and negatives, we explore the relationship between certain web pages being browsed and the status of the browser (synced/not synced). Specifically, there are two different cases that our algorithm incorrectly detects as synced.

- This case is depicted in Figure 6 (Appendix A). A user who owns two different devices, which are not synced, visits the same web page at two

TABLE 1: Representation of all the values extracted from the results with two attributes.

Results with two attributes							
Type of data	Total number of data	True positives	True negatives	False positives	False negatives	Total true guesses	Total false guesses
Numeric	84,070	41,328	12,225	25,059	5,458	53,553	30,517
Percentage	100%	49,2%	14,5%	29,8%	6,5%	64%	36%

different timestamps. The first visit initializes a URL path U_1 to U_2 that ends up to U_{target} at time t_i. Later, at time t_k, the user re-visits U_{target} from another device, but without being synced. This is the case where the user may have copied the URL link to return to the web page. Our algorithm will mark those requests as synced, while they are not. Notice, that this case represents an *implicit syncing*.

- This case is depicted in Figure 7 (Appendix A). Assume we have two different users, Alice and Bob, who each own two devices, A_1 and A_2, B_1, and B_2, respectively. Both users have their devices synchronized to their browser account. Alice visits a specific web page, W_{target}, using device A_1 at a timestamp t_i, and then she saves that web page to return later. Then, she returns at a timestamp t_k to the web page by accessing it through the synchronization feature. If Bob also visits W_{target} at a different timestamp t_z, using device B_1, and then returns to it utilizing the synchronization feature from device B_2, then our approach will not be able to distinguish which requests have been issued from which user. However, there are some cases where our approach will pair the requests from A_1 and A_2 and B_1 and B_2, correctly.
  - i) If $t_i == t_z$, then our approach will pair correctly the requests of A_1 and A_2, and the requests of B_1 and B_2, as synced. This will happen because the algorithm searches for requests that are issued at different timestamps. Thus, if $t_i == t_z$, after the algorithm (lines 11-14 of Algorithm 1) groups them all together because of the same URL, it should separate them at the next step because they have the same timestamp.
  - ii) Similarly, if $t_k == t_n$ and $t_i \neq t_z$, the requests that are issued from A_1, A_2 and B_1, B_2 will be paired correctly, because they will be stored on different records of the `different_timestamps` list (lines 11-14 of Algorithm 1).

  This false-positive category can be also generated for users with multiple devices (more than two). For instance, we assume that we have two users Alice and Bob, but now Alice owns three devices instead of two, A_1, A_2, A_3 and Bob owns two devices, B_1 and B_2. Both users have their devices synced to their browser account. Alice visits a specific web page, W_{target} at a timestamp t_i, using device A_1 and then saves this web page to return later. Then, she returns to the web page using both devices A_2, and A_3 at different timestamps t_k and t_m. Bob also visits W_{target} at a timestamp t_z, using device B_1, and re-visits W_{target} using device B_2, utilizing the synchronization feature, at a different timestamp t_n. In this scenario, our methodology will not be able to distinguish which requests have been performed from which user. Similarly to the scenario where the users own two devices, the cases that our approach will pair the requests from A_1, A_2, A_3 and B_1 and B_2 correctly are the following:
  - i) If $t_i == t_z$, then our approach will pair correctly the requests of A_1, A_2, and A_3 and the requests of B_1 and B_2, as synced.
  - ii) If $t_i \neq t_z$, but either $t_k == t_n$ *or* $t_m == tn$, then again the requests will be paired at the correct synced lists.

6.5. Discussion

As mentioned in Section 3.1, where we define the threat model of our approach, we only use the attributes that can be collected passively by the web site in the *least* intrusive way. However, additional attributes, such as canvas, screen resolution, list of plugins, use of local storage, WebGL features, list of fonts and color depth, could be also collected through more intrusive JavaScript-driven injections. In the case where we collect both passively and actively attributes, the accuracy of our algorithm could be highly improved. For instance, canvas-related fingerprinting that works by exploiting the HTML5 canvas element could result in highly unique fingerprints. Therefore, we assume that if we use more intrusive JavaScript-driven injection for gathering the attributes mentioned above, the accuracy of our methodology can be potentially improved. However, our intention is not to explore such an aggressive fingerprinting threat model, which has been already explored in many other papers [23, 24, 25, 26, 27].

Another issue we consider is if we can use more attributes for better distinguishing machine-generated from user-generated requests in order to further reduce false positives. We incorporate two more attributes after the execution of the similarity function for cookies (after line 36 of Algorithm 1). Before appending the requests to the `final_result` list we identify which requests have also the same IP address and user agent, additionally to cookies. We compare the user agent and IP address values for each list that contains the same URL, different timestamps and similar cookies (as defined in the similarity function, lines 43-61 of Algorithm 1). We append only the requests that their user agent and IP address values are also the same to the `final_result` list. Table 3 presents the quantitative results of our approach when considering all five attributes we collect, namely URL, timestamp, user agent, IP address and cookies. We can observe that the user agent and the IP address attributes decrease the success rates of the algorithm instead of increasing its accuracy. More specifically, the true guesses reach only the 57% and the false guesses reach the 43%. There is

TABLE 2: Representation of all the values extracted from the results with three attributes.

Results with three attributes							
Type of data	Total number of data	True positives	True negatives	False positives	False negatives	Total true guesses	Total false guesses
Numeric	138,962	57,661	48,759	15,636	16,906	106,420	35,542
Percentage	100%	41,5%	35,1%	11,2%	12,2%	76,6%	23,4%

a large increase in the false negatives proportion, which grows to 28% from 12% (three attributes).

The user agent attribute is not useful because it takes very distinct values. Most of the times, if the browser and the platform are the same, the user agent attribute is also the same. In our case, the browser section is the same for all users, because we collect the data through a Mozilla extension. The only part that differs is the platform. The platform options are also very specific. However, there is a proportion of requests that are removed from the `possible_synced` list, but they should not. Those are the requests that are marked correctly as synced from the previous steps, and they have been performed by the same user, from two different devices that use different platforms. This means, that their user agent field differs and they are removed when comparing the user agent field, while they should not. Those requests increase the false negative percentage.

Last but not least, as far as the attribute of the IP address is concerned, it is a value that constantly changes. Thus, it cannot be used to correlate sessions of different timestamps, because the IP addresses of those sessions will be totally different. We want to identify synced sessions that have been performed by the same user through multiple devices, however, the IP address of these devices is likely to be different. This can further drive a rise in false negatives.

7. Related Work

Peter Eckersley conducts the first study about browser fingerprinting in 2010 [1] and the findings show that 83,6% of the fingerprints are extremely unique [1], meaning that those fingerprints could differentiate the users. Browser fingerprinting is divided into many different fields and the research in that area is broad and extensive. Although our work is orthogonal to the already existing literature about browser fingerprinting, our approach is associated with the broad field of cross-device tracking. However, we are the first to investigate the use of browser fingerprinting together with synchronized browsers. The existing literature about browser fingerprinting and cross-device tracking is described in the sections below.

Upathilake et al. [28] perform a systematic classification of browser fingerprinting techniques. Laperdix et al. [2] publish a survey about browser fingerprinting, and they list all the research performed in the domain of browser fingerprinting. Furthermore, they discuss the current state of browser fingerprinting and the challenges that occur from it. Apart from the traditional and known ways to fingerprint the users, there are some novel tracking mechanisms that have been proposed recently [4, 7, 5, 6].

A lot of detection, mitigation and prevention mechanisms have been proposed by the research community over the years. Nikiforakis et al. propose PriVaricator [29]. The authors focus on the linkability of browser fingerprint and not to its uniqueness. Wu et al. [30] propose UNIGL. Their work focuses on the WebGL feature, that is responsible of rendering WebGL tasks. Bird et al. [31] propose a semi-supervised learning approach for detecting fingerprinting scripts. More approaches based on Machine Learning, proposed by Iqbal et al. [32], Bahrami et al. [33], and Rizzo et al. [34]. Moreover, Durey et al. [35] propose an algorithm that detects browser fingerprinting scripts. Their algorithm is based on an incremental process and it relies on both automatic and manual decisions. Furthermore, Moad et al. [36] introduce the Fingerprint Defender, which is a Google Chrome extension that anonymize user artifacts for tracking.

Due to the wide research around defensive techniques against browser fingerprinting for tracking users, some researchers focus on the evaluation of the effectiveness of those techniques. Acar et al. [23] perform a large-scale study for three tracking mechanisms and their results show that even sophisticated and privacy-aware users face great difficulties in evading tracking mechanisms. Moreover, there are studies that explore and evaluate the effectiveness of browser fingerprinting. Gómez-Boix et al. [37] conduct a study to explore the effectiveness of browser fingerprinting at uniquely identifying a large group of users when analyzing millions of fingerprints over a few months. In addition, Laperdix et al. [16] explore the validity of browser fingerprinting at the time (2016). More recently, Fietkau et al. [38] do a systematic analysis of various fingerprinting tools.

A work by Vastel et al. [39] studies the browser fingerprinting from a different perspective. They study browser fingerprinting as crawler detection mechanism. Furthermore, Andriamilanto et al. [40] use browser fingerprinting as web authentication mechanism.

7.1. Cross-device tracking

Cross-device tracking (CDT) comprises of a set of technologies and methods that are used to track users across multiple devices, such as mobile devices, desktop computers, laptops, and connected TVs [12]. It works by matching activity across different devices to the same user who performs them by using shared identifiers, that are in use on each user's device [8]. Cross-device tracking could be deterministic or probabilistic. Deterministic CDT utilizes 1st-party login services that require user authentication, while in probabilistic CDT 3rd parties attempt to identify which devices belong to the same user based on network access data, and common behavioral patterns in browsing history [12]. The existing literature about cross-device tracking is reported in the rest of this section.

Brookman et al. [8] and Zimmeck et al. [13] conducted studies to assess what information about CDT is observable from the perspective of the end-user, and evaluate the privacy implications of ML applications to cross-device data, respectively. Mavroudis et al. [10] and

TABLE 3: Representation of all the values extracted from the results with five attributes.

Results with five attributes							
Type of data	Total number of data	True positives	True negatives	False positives	False negatives	Total true guesses	Total false guesses
Numeric	138,962	44,112	35,031	20,455	39,364	69,143	69,819
Percentage	100%	31,7%	25,3%	14,7%	28,3%	57%	43%

Arp et al. [14] explores the security and privacy implications of ultrasound based technologies when used for CDT. Korolova et al. [9] showed that cross-app tracking is feasible using nearby BLE. Moghaddam et al. [11] examined the advertising and tracking ecosystem of OTT streaming devices. Finally, Solomos et al. [12] designed Talon, a practical framework for CDT measurements.

8. Conclusion

In this paper we are the first, to the best of our knowledge, to deliver a framework that can be used by web sites to detect synchronized HTTP requests, issued by the same user from browsers on different devices. For detecting this, we reconstruct different sessions based on their requested attributes and timestamps. We evaluated our methodology through a user study we conduct, with 30 users in total. We collected 138,962 HTTP requests, with a browser extension we produced for Mozilla Firefox, and the $\sim$77% of all the sessions are detected correctly from our algorithm. Our results show that the detection of synchronized sessions through browser fingerprinting is possible with a high success rate. They also indicate major implications to user privacy, that have not been investigated before.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. The research conducted in this paper was triggered by the project "Collaborative, Multi-modal and Agile Professional Cybersecurity Training Program for a Skilled Workforce In the European Digital Single Market and Industries" (CyberSecPro) project. This project has received funding from the European Union's Digital Europe Programme (DEP) programme under grant agreement No 101083594. The authors are grateful for the financial support of this project that has received funding from the European Commission. The views expressed in this paper represent only the views of the authors and not of the European Commission or the partners in the above-mentioned project. Moreover, the work of this paper was also supported by the European Union's Horizon 2020 and Horizon Europe research and innovation programmes under grant agreements No.101070599 (SecOPERA), No. 830929 (CyberSec4Europe) and No. 101007673 (RESPECT).

References

[1] P. Eckersley, "How unique is your web browser?" in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2010, pp. 1–18.

[2] P. Laperdrix, N. Bielova, B. Baudry, and G. Avoine, "Browser fingerprinting: A survey," *ACM Transactions on the Web (TWEB)*, vol. 14, no. 2, pp. 1–33, 2020.

[3] T. Bujlow, V. Carela-Español, J. Sole-Pareta, and P. Barlet-Ros, "A survey on web tracking: Mechanisms, implications, and defenses," *Proceedings of the IEEE*, vol. 105, no. 8, pp. 1476–1510, 2017.

[4] K. Solomos, J. Kristoff, C. Kanich, and J. Polakis, "Tales of favicons and caches: Persistent tracking in modern browsers," in *Network and Distributed System Security Symposium*, 2021.

[5] P. Laperdrix, O. Starov, Q. Chen, A. Kapravelos, and N. Nikiforakis, "Fingerprinting in style: Detecting browser extensions via injected style sheets," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2507–2524.

[6] T. Wu, Y. Song, F. Zhang, S. Gao, and B. Chen, "My site knows where you are: A novel browser fingerprint to track user position," in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.

[7] I. Fouad, C. Santos, A. Legout, and N. Bielova, "Did I delete my cookies? Cookies respawning with browser fingerprinting," *arXiv preprint arXiv:2105.04381*, 2021.

[8] J. Brookman, P. Rouge, A. Alva, and C. Yeung, "Cross-device tracking: Measurement and disclosures." *Proc. Priv. Enhancing Technol.*, vol. 2017, no. 2, pp. 133–148, 2017.

[9] A. Korolova and V. Sharma, "Cross-app tracking via nearby bluetooth low energy devices," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, 2018, pp. 43–52.

[10] V. Mavroudis, S. Hao, Y. Fratantonio, F. Maggi, C. Kruegel, and G. Vigna, "On the privacy and security of the ultrasound ecosystem," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 2, pp. 95–112, 2017.

[11] H. Mohajeri Moghaddam, G. Acar, B. Burgess, A. Mathur, D. Y. Huang, N. Feamster, E. W. Felten, P. Mittal, and A. Narayanan, "Watching you watch: The tracking ecosystem of over-the-top tv streaming devices," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 131–147.

[12] K. Solomos, P. Ilia, S. Ioannidis, and N. Kourtellis, "Talon: An automated framework for cross-device tracking detection," in *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, 2019, pp. 227–241.

[13] S. Zimmeck, J. S. Li, H. Kim, S. M. Bellovin, and T. Jebara, "A privacy analysis of cross-device tracking," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1391–1408.

[14] D. Arp, E. Quiring, C. Wressnegger, and K. Rieck,

"Privacy threats through ultrasonic side channels on mobile devices," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2017, pp. 35–47.

[15] StatCounter, 2022, Last accessed 13 September 2022. [Online]. Available: https://gs.statcounter.com/

[16] P. Laperdrix, W. Rudametkin, and B. Baudry, "Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 878–894.

[17] Google, Last accessed 28 Feb 2022. [Online]. Available: https://www.google.com/chrome/browser-features/

[18] Mozila Firefox, Last accessed 28 Feb 2022. [Online]. Available: https://www.mozilla.org/en-US/firefox/sync/

[19] A. Paverd, A. Martin, and I. Brown, "Modelling and automatically analysing privacy properties for honest-but-curious adversaries," *Tech. Rep*, 2014.

[20] Statista, "User population of selected internet browsers worldwide from 2014 to 2021," 2022, Last accessed 12 May 2022. [Online]. Available: https://www.statista.com/statistics/543218/worldwide-internet-users-by-browser

[21] Mozilla Developer Networks, Last accessed 29 March 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Anatomy_of_a_WebExtension

[22] MDN, Last accessed 18 May 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest/onBeforeRequest

[23] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The web never forgets: Persistent tracking mechanisms in the wild," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 674–689.

[24] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel, "Fpdetective: dusting the web for fingerprinters," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 1129–1140.

[25] N. M. Al-Fannah, W. Li, and C. J. Mitchell, "Beyond cookie monster amnesia: Real world persistent online tracking," in *Information Security: 21st International Conference, ISC 2018, Guildford, UK, September 9–12, 2018, Proceedings 21*. Springer, 2018, pp. 481–501.

[26] 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.

[27] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 541–555.

[28] R. Upathilake, Y. Li, and A. Matrawy, "A classification of web browser fingerprinting techniques," in *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, 2015, pp. 1–5.

[29] N. Nikiforakis, W. Joosen, and B. Livshits, "Privaricator: Deceiving fingerprinters with little white lies," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 820–830.

[30] S. Wu, S. Li, Y. Cao, and N. Wang, "Rendered private: Making {GLSL} execution uniform to prevent {WebGL-based} browser fingerprinting," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1645–1660.

[31] S. Bird, V. Mishra, S. Englehardt, R. Willoughby, D. Zeber, W. Rudametkin, and M. Lopatka, "Actions speak louder than words: Semi-supervised learning for browser fingerprinting detection," *arXiv preprint arXiv:2003.04463*, 2020.

[32] U. Iqbal, S. Englehardt, and Z. Shafiq, "Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1143–1161.

[33] P. N. Bahrami, U. Iqbal, and Z. Shafiq, "Fp-radar: Longitudinal measurement and early detection of browser fingerprinting," *arXiv preprint arXiv:2112.01662*, 2021.

[34] V. Rizzo, S. Traverso, and M. Mellia, "Unveiling web fingerprinting in the wild via code mining and machine learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 1, pp. 43–63, 2021.

[35] A. Durey, P. Laperdrix, W. Rudametkin, and R. Rouvoy, "An iterative technique to identify browser fingerprinting scripts," *arXiv preprint arXiv:2103.00590*, 2021.

[36] D. Moad, V. Sihag, G. Choudhary, D. G. Duguma, and I. You, "Fingerprint defender: Defense against browser-based user tracking," in *International Symposium on Mobile Internet Security*. Springer, 2021, pp. 236–247.

[37] 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.

[38] J. Fietkau, K. Thimmaraju, F. Kybranz, S. Neef, and J.-P. Seifert, "The elephant in the background: A quantitative approachto empower users against web browser fingerprinting," in *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*, 2021, pp. 167–180.

[39] A. Vastel, W. Rudametkin, R. Rouvoy, and X. Blanc, "Fp-crawlers: studying the resilience of browser fingerprinting to block crawlers," in *MADWeb'20-NDSS Workshop on Measurements, Attacks, and Defenses for the Web*, 2020.

[40] N. Andriamilanto and T. Allard, "Brfast: a tool to select browser fingerprinting attributes for web authentication according to a usability-security trade-off," in *Companion Proceedings of the Web Conference 2021*, 2021, pp. 701–704.

A. False positives cases

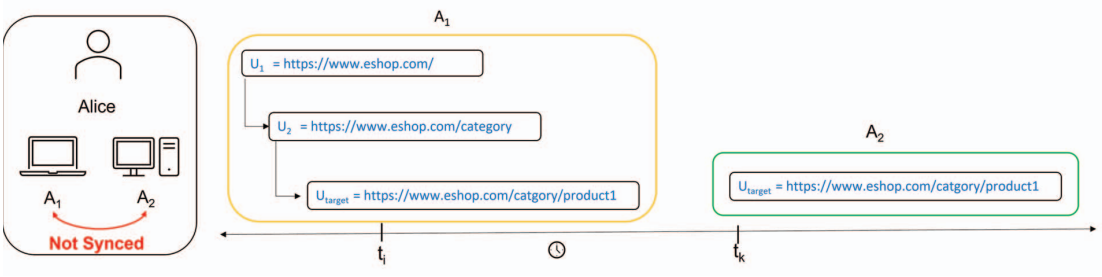

Figure 6: **No synchronization.** A user who owns two different devices, which are not synced, visits the same web page at two different timestamps. The first visit initialized a URL path U_1 to U_2 that ends-up to U_{target}. Later, the user re-visits U_{target} from another device, but without being synced. This is the case where the user may copied the URL link. Our algorithm will mark those requests as synced, while they are not.

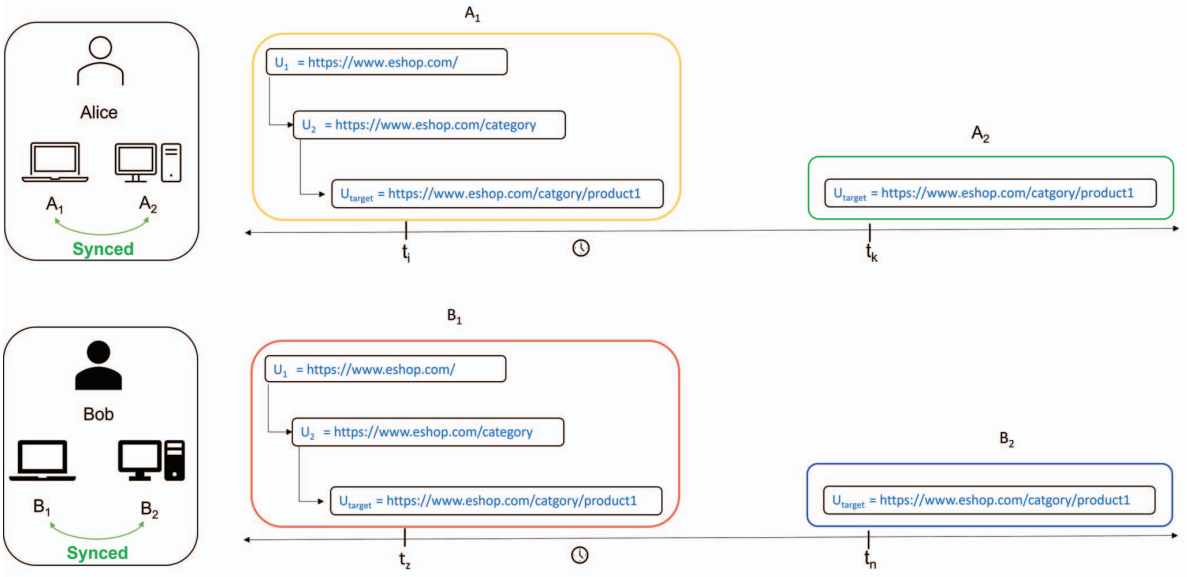

Figure 7: **Wrong synchronization pairing.** Two users, Alice and Bob, own devices A_1, A_2, and B_1, B_2, respectively, which are synchronized. Alice visits a specific web page W_{target} following a URL path from the landing page of the web site at time t_i, from device A_1. At a later time, t_k, Alice re-visits W_{target}, from device A_2. However, this time she does not initiate any of the intermediate requests, because she accesses W_{target} through the synchronization feature. Bob also visits the same web page W_{target} following a URL path from the landing page of the web site at time t_z, from device B_1. At a later time, t_n, he re-visits W_{target}, from device B_2. Our methodology could not distinguish which requests have been performed form Alice's devices and which from Bob's. However, there are some cases where the algorithm pairs the requests correctly, and they are described in Section 6.4.

B. Example of Synced field in the database

```
ID:1961
URL:https://nsi.activity.api.bbc.com/my/plays
Timestamp: 18/4/2022, 7:33:55 PM
User_Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
IP:  194.42.16.139
Cookies: Cookie Domain: .cnn.com .....
Synced: [973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985]


ID: 1966
URL: https://bbc.com/templates/data/datetime.aspx
Timestamp: 17/4/2022, 12:09:04 AM
User_Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:97.0),
Gecko/20100101 Firefox/97.0
IP:  62.228.18.230
Cookies: Domain: bbc.com  ....
Synced:  NULL
```

Figure 8: **Example of the synced field in the database.** The figure above shows an instance of the reconstruction database. The synced field could contain either `null` values or a list with IDs, that indicate the requests that are also synced with the specific request.