Beyond Black and White: Combining the Benefits of Regular and Incognito Browsing Modes

John Korniotakis[1], Panagiotis Papadopoulos[2] and Evangelos P. Markatos[1,2]
[1] *University of Crete, Greece*
[2] *FORTH, Greece*

Keywords: Incognito Mode, Web Browsers, Web Cookies, User Privacy.

Abstract: As an increasing number of users seem to be concerned about the sophisticated tracking approaches that web sites employ, most modern browsers provide a privacy-preserving browsing mode. This so called "Incognito" mode enables users to browse web sites with sensitive (e.g., medical-, religious-, and substance-abuse-related) content by providing them with a clean-state and disposable browser session. Although incognito-mode browsing is useful, users will eventually need to switch back to regular web browsing as they need to log in to their favorite web sites (e.g., Gmail, Facebook, etc.). However, whenever they want to access another unfamiliar or sensitive web site, they are forced to switch back to Incognito mode, and so on and so forth. Pretty soon, users find themselves switching all the time back and forth between regular and incognito mode. Unfortunately, such a chain of actions, is not only tiresome, but may also turn out to be error-prone as well, since users may accidentally use regular browsing mode to visit web sites they intended to access in incognito mode. To provide users with a convenient and privacy-preserving browsing experience, in this paper we propose GRISEO: a new browsing mode that aims to act as a middle-ground, thus enabling users to get the best of both words: the privacy of incognito mode along with the convenience of the regular browsing mode. Our approach is founded on a whitelist-based solution where users "whitelist" the sites they trust and from which they are willing to receive cookies that will persist even after a single browsing section is over. The rest of the sites are considered black-listed and are allowed to plant only ephemeral cookies: cookies that will be deleted at the end of the browsing session. Our preliminary performance results of our prototype, show that for the average web page access, the rendering time while browsing in GRISEO mode is just 5.9% more than while in Chrome's Regular Browsing mode. We believe that the overhead of GRISEO is very small and it will be hardly noticeable by most users.

1 INTRODUCTION

Over the last few years, an increasing amount of our every actions are being conducted online: we read news, we watch movies, we communicate with friends, we make payments, we purchase goods, we monitor our health - everything online. At the same time, we have started to realize that an increasing amount of web entities, such as advertisers (Papadopoulos et al., 2018a; Bashir and Wilson, 2018; Papadopoulos et al., 2017b) and various third party trackers (Roesner et al., 2012; Papadopoulos et al., 2017a; Agarwal et al., 2020), collect and correlate all information they can get their hands on.

To help users preserve some of their privacy online, most popular web browsers provide an "Incognito" (or "private") browsing mode. In this mode, browsers refrain from either providing or permanently storing *any state* that may allow trackers to re-identify the user across different browsing sessions.

To be more particular, this Incognito mode does not reuse any cookies stored previously during the regular browsing mode. Thus, even if the user has visited the very same website only a few minutes ago in regular mode, while in Incognito, the website cannot retrieve any cookies from this past visit. Instead, the remote web server acts as if the user landed on the website for the first time (unless dubious hardware fingerprinting techniques are deployed (Cao et al., 2017; Klein and Pinkas, 2019)).

What is more, even if the web browser does accept new cookies to allow a given web server to identify the recipient of the transmitted HTTP requests, their lifetime is closely-associated with the lifetime of the Incognito mode; as soon as the Incognito browsing mode is over, the browser discards all cookies received. As a consequence, any future visits to the same web site will not be able to provide the cookies of the current visit.

In this aspect, each browsing session is "dispos-

Korniotakis, J., Papadopoulos, P. and Markatos, E.
Beyond Black and White: Combining the Benefits of Regular and Incognito Browsing Modes.
DOI: 10.5220/0009885301920200
In *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications (ICETE 2020) - SECRYPT*, pages 192-200
ISBN: 978-989-758-446-6

able" and self-contained. It does not interact with any past or future sessions: each browsing session (i) does not have access to any previously-stored cookies, and (ii) can not provide any cookies to future sessions. Additionally, when browsing in Incognito mode, the visited websites are not part of the catalog that includes the user's browsing history. Apart from the cookies that will get purged when the Incognito mode is over, any information in the Service Workers(Google Developers, 2020) will be deleted and unregistered respectively.

Because of all those privacy-preserving features, incognito browsing has started to become popular with users who would like to access sensitive information including medical issues, sexual, political or religious preferences, substance-abuse, etc.

After browsing for a while in incognito mode, users frequently realize that they need to switch back to regular browsing mode in order to access their logged in accounts (by providing their previously stored cookies) in sites such as Gmail, Twitter, or Facebook. However, when they want to access sensitive information again, they need to switch back to Incognito mode, and so on and so forth, leading to an endless cycle between Incognito and regular browsing sessions and multiple open browser windows.

Unfortunately, this frequent back-and-forth between Incognito and regular browsing modes has two main drawbacks: (i) it is not convenient at all, and (ii) it may lead to errors as users may access sensitive information by mistake while browsing in regular mode, thus risking their privacy.

To help the incognito browsing mode reach its full potential, we propose GRISEO: an approach to get the best of both worlds: (i) the convenience of the regular browsing mode and (ii) the privacy of the Incognito mode. More specifically, in GRISEO, we enhance the Incognito mode by pre-loading state from sessions that took place via the regular mode for specific white-listed sites the the user trusts and in which they want to remain logged in. Thus, for these white-listed sites, the browser (i) retrieves their previously-stored cookies and (ii) stores (for future browsing sessions) any more cookies received - much like it would do in the regular browsing mode. This concurrent browsing in two modes (regular and incognito) may remind the reader of Firefox's containers where each site is assigned to a container (type) and sites do not have access to cookies outside their container.

Although it may sound simple, things get complicated when websites of white-listed domains are included in other (let us call them "black-listed") domains. Indeed, in cases of a website of a white-listed domain that includes a widget of a black-listed domain the policy play field starts to get bumpy. Things may get even more complicated when we consider interactions with more complex mechanisms including cookie synchronization. In the rest of the paper we will explain how we negotiate these complicated interactions between black and white-listed domains.

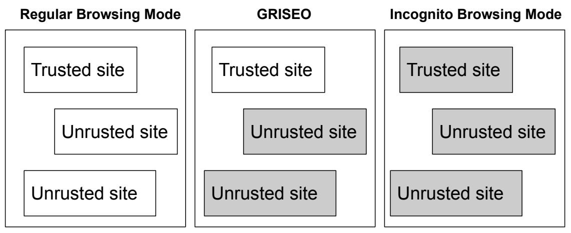

Figure 1: GRISEO works as a middle-ground between the regular and Incognito browsing mode, allowing users to define which sites can be considered as trusted and which cannot.

In summary, this paper makes the following contributions:

1. We design GRISEO: a mechanism to automatically log the user into a web site they trust while in Incognito browsing mode.
2. We implement a prototype of our approach as an extension for the popular web browser of Google Chrome.
3. We conduct a preliminary evaluation of our prototype's performance with the results showing that GRISEO adds no more than 5.9% milliseconds of latency for the average web page access.

2 SYSTEM DESIGN

In this section, we describe the details of our approach. GRISEO, as depicted in Figure 1, is a browsing mode that acts as a middle ground between (i) regular browsing mode, where all web sites are considered trusted and (ii) the Incognito mode where all web sites are considered untrusted. In GRISEO, the user can define what sites they trust and by doing so, they can fully enjoy the provided user experience. As a result, the browser keeps the states of these sites across the different Incognito browsing sessions.

To achieve the above, in our system, we assume that the user has divided the web sites into two sets:

1. The *white-listed sites*: these are the sites that the user is willing to *accept* (and permanently store) cookies from and to *send* (previously stored) cookies to. These are usually sites that the user visits frequently, and quite possibly, sites the user has an account for. Such sites may include the

user's email server, the user's social media, the user's bank, etc.

2. The *black-listed sites*: these are the sites the user prefers to access in Incognito mode. Such sites may include sites that provide medical or religious information, etc. Although a plethora of other sites, which are neither white-listed, nor sensitive, may also exist , we consider all of them as black-listed.

We should make it clear that the user does not really divide the sites in two sets: the user just white-lists some sites, and the rest are implicitly black-listed.

In GRISEO, whenever the user starts a browser, it starts in Incognito mode by default. Then:

- If a user accesses a black-listed web site, the browser keeps on running in Incognito mode.

- If the user accesses a white-listed web site, the user treats this access as if it was in regular browsing mode and: (i) the stored cookies are retrieved from the local storage, (ii) these (previously stored) cookies are provided to the web site as expected, and, (iii) at the end of the browsing session, all cookies received during this session are stored, so that they can be used in future accesses to this web site in subsequent browsing sessions.

2.1 Protecting White-listed Cookies in Incognito Mode

The design so far seems relatively simple: white-listed web sites get all their cookie history, while black-listed web sites only have ephemeral cookies: cookies that last only for the current browsing session. Unfortunately, things are much more complicated than they seem at a first glance. Indeed, cookies can interact in very sophisticated ways, one of which is *cookie synchronization* (Acar et al., 2014; Papadopoulos et al., 2019; Papadopoulos et al., 2018b). Before explaining how cookie synchronization complicates our GRISEO system, let us try to understand how cookie synchronization works.

Let us assume a user is browsing several domains such as `website1.com` and `website2.com`, in which there are third party domain widgets (such as `tracker.com` and `advertiser.com`). As a result, these two third party sites have the chance to set their own cookies in the user's browser, in order to re-identify the user in the future.

In these cookies, each domain assigns an ID to each user. This way, `tracker.com` knows the user by the ID `user123`, and `advertiser.com` knows the same user by the ID `userABC`: note that the two different domains use different IDs for the same user - they do not know that these two different IDs correspond to the same user.

Now let us assume that the user lands on a website (say `website3.com`), which includes some JavaScript code from `tracker.com` but **not** from `advertiser.com`. Thus, since `advertiser.com` is not included in `website3.com`, it does not (and cannot) know which users visit `website3.com`. Let us assume now that as soon as the code of `tracker.com` is called, a `GET` request is issued by the browser to `tracker.com`, which responds back with a REDIRECT request, instructing the user's browser to issue another GET request to `advertiser.com` this time, using a specifically crafted URL:

GET advertiser.com?syncID=**user123**&publisher=**website3.com**
Cookie: {cookie_ID=**userABC**}

When `advertiser.com` receives the above request along with the cookie ID `userABC`, it can easily find out that `userABC` visited `website3.com`. To make matters worse, `advertiser.com` also finds out that the user whom `tracker.com` knows as `user123`, and the user `userABC` is basically one and the same user. Effectively, this cookie synchronization enabled `advertiser.com` to collaborate with `tracker.com`, in order to find out that user `userABC` and `user123` is one and only one user.

Let us now focus on what is the effect of cookie synchronization when a black-listed web site *A* includes the widget of a white-listed web site *B*. In this case, *A* (since it is black-listed) knows its visitors by a unique (and disposable) pseudonym. However, web site *B* (which is white-listed) knows its visitors by their real name (if they have an account with *B*).

Now let us consider what will happen if *B* engages in cookie synchronization with *A*. In that case, *B* (which knows the users by their real name) will synchronize cookies with *A*. As a result, *A* will now know its visitors by their real name. Thus, *A*, which is black-listed, will be able to really know the users by their real name and follow them through the Internet. One might think that since *A* is black-listed this knowledge is somehow ephemeral and once the browser is closed, site *A* will forget all information about the users and their names.

Make no mistake: each and every time the user visits *A*, *A* will know the user by their real name: the cookie synchronization that *B* did will act as a permanent cookie for site *A*. At this point it is natural for the reader to wonder: *How did this happen? How can a black-listed site know a user by their real name?* This happened because a black-listed web site included a widget of a white-listed one, and because

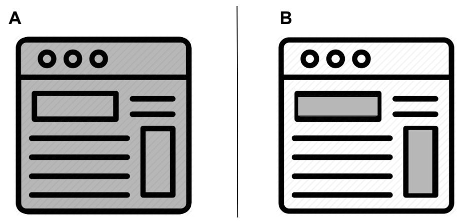

Figure 2: The different policies followed by GRISEO in case of mashup websites: (A) if the website is black-listed, all components in the web page will behave as black-listed, (B) if the website is white-listed, all components in the web page will behave as their domain of origin has been individually defined.

the white-listed one engaged in cookie synchronization with the black-listed site. To cut a long story short, allowing interactions between black-listed and white-listed sites has resulted in information leakage. One can find more examples of undesirable behaviour when a white-listed site is enclosed in a black-listed one, and due to their interaction the white-listed site provides information to the black-listed one: information which the user did not expected to give out.

2.2 Policy

To be able to explain the policy that we need to follow, we define the *context* within which each web request is made. The context is the web page within which a web request is made. For example, let us assume that we access the web page of site *A*. Let us also assume that the HTML code of *A* includes a request to an image of web site *B*. Then, the request to this image is issued within the context of *A*. If *A* is white-listed, then the context of the request is a white-listed one. Otherwise, it is a black-listed one. If the web request is the one on the address bar of the browser, we say that this is the *top-level* context. As depicted in Figure 2, our policy is defined as follows:

- A web page request that is issued inside a black-listed context is always treated as black-listed.
- A web page request that is issued inside a white-listed context will behave as it has been defined: white-listed ones will behave as white-listed, and black-listed ones will behave as black-listed.
- A web page request that is issued at the top level will behave as it has been defined: white-listed ones will behave as white-listed, and black-listed ones will behave as black-listed.

3 SYSTEM IMPLEMENTATION

To facilitate the easy deployment of our approach, GRISEO is implemented in JavaScript as a browser extension. For rapid prototyping, our implementation has focused on the Chrome browser, although it can be easily extended to support other browsers, such as Firefox. The total implementation consists of no more than 600 lines of code. At this point we must say that the Firefox implementation (on top of existing containers[1]) might have been slightly faster/easier. However, we chose to do first an implementation in Chrome, as it is most widely used and we could reach a much higher end-user community. Indeed, recent statistics suggest that Chrome is used by close to 60% of the users, while Firefox is being used by a bit less than 6% [2].

To place our work in the proper context, we consider the following browsing modes:

1. **The User Accesses a Website in Regular Browsing Mode.** In these cases we do nothing. This is a regular browsing mode that we do not interfere with.
2. **The User Accesses a Black-listed Site in Incognito Browsing Mode.** In this case we also do nothing. The user wants to browse the web site in Incognito mode - an operation which is fully supported by Chrome without our intervention.
3. **The User Accesses a White-listed Web Site in Incognito Browsing Mode.** This is exactly where the GRISEO functionality kicks in. In the rest of this section we explain how we handle this browsing situation and enable the user to provide cookies (for example to log in the web site) without leaving incognito mode.

A. Page Visit: As depicted in the workflow of Figure 3, as soon as a user visits a web page in their browser, the `chrome.webRequest.onBeforeRequest()` handler is invoked. The first thing the function does is to block the request and then, it searches the local storage to find whether the website is white-listed. To do so, the function consults the `whitelistedWebsites` object. If the website is found there, then it means that the site is white-listed and the extension does two things:

- It adds the tab that the site was visited in, in the site's regular or Incognito tabs array (depending what mode the user is browsing on).

[1] https://addons.mozilla.org/el/firefox/addon/multi-account-containers/

[2] https://www.w3counter.com/globalstats.php

- Searches for (previously) stored cookies of this website by looking up the `cookies` object in the local storage. If it finds any cookies, the extension checks whether the browsing mode is regular or Incognito (by checking the `chrome.inIncognitoContext` value). If the browsing mode is Incognito, we load the cookies in the cookie store with id 1, otherwise (we are in regular browsing mode) we load the cookies in in the cookie store with id 0. Cookies are loaded by invoking the function `chrome.cookies.set()`. Obviously, if there are no cookies for this website (such as when the user accesses this site for the first time), no cookies are loaded. After that, the event handler redirects back to the site that was initially visited, and the cookies are used immediately. In addition to the above handler, which is invoked on page visit, there are a few more handlers which handle various events including (i) tab creation, (ii) tab closing, (iii) attachment to a window, (iv) detachment from a window, and (v) window closing. We will describe the functionality of GRISEO for these events below.

B. Browser Tab Closing: When a browser tab is closed, the `chrome.tabs.onRemoved()` function is invoked which (i) removes the tab from the proper window id in the global `windows` object, and then, (ii) removes the tab from the tab array of the site. If this was the last tab of the site, the extension gets all the cookies of this site using `chrome.cookies.get()`. Then it removes them from the cookie store by using `chrome. cookies.remove()`, and stores them in the proper field of the `cookies` object in the local storage.

C. Browser Tab Creation: When a tab is created, the `chrome.tabs.onCreated()` function is invoked to add the newly created tab's id to the corresponding window id in the `windows` global object.

D. Browser Tab Attachment - Detachment: When a tab is detached from a window (say `A`) and attached to another window (say `B`), the `chrome.tabs.onDetatched()` function is called to remove id of this tab from window `A` and the `chrome.tabs.onAttatched()` function is called to add the id of this tab to window `B`.

E. Browser Window Closing: When a window is closed, we cannot get the ids of the tabs that were open in this window anymore, because it is now closed. For that reason we are using the global `windows` object, to keep track of the tabs in each window. So when a window closes, the `chrome.windows.onRemoved()` function is invoked to check the tab ids of the window id that is closing and perform the necessary final clean-up activities.

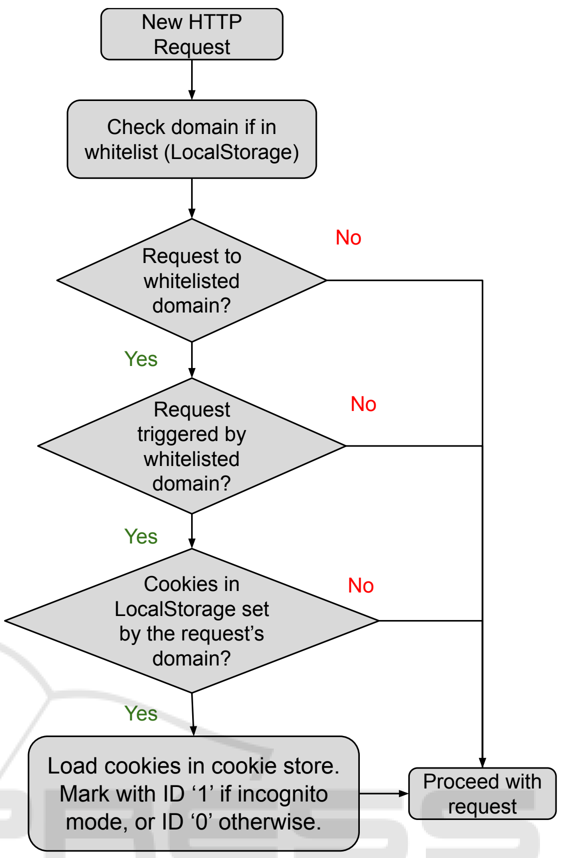
New HTTP Request
Check domain if in whitelist (LocalStorage)
Request to whitelisted domain?
No
Yes
Request triggered by whitelisted domain?
No
Yes
Cookies in LocalStorage set by the request's domain?
No
Yes
Load cookies in cookie store. Mark with ID '1' if incognito mode, or ID '0' otherwise.
Proceed with request

Figure 3: High-level overview of our prototype's workflow during each newly generated HTTP request of the user's browser.

F. White-listing a Website: The user can white-list a website by clicking the extension icon and then the button "White-list this website". Then, the extension will store this web site in the `whitelistedWebsites` object in the local storage. All the cookies of that site are stored in the `cookies` object in the local storage. Afterwards, the extension finds all the tabs that the website is open in using the `chrome.tabs.query()` function, and then creates an entry in the `whitelistedWebsites` object in the local storage with all the tabs that it was open in (either in Incognito or regular browsing mode). Then, the extension gets all the cookies currently loaded for this website using `chrome.cookies.get()` and stores them in the `cookies` object in the local storage. The user is also able to *remove* a site from the set of white-listed sites using the `remove` button. When that button is clicked, the website is removed from the `whitelistedWebsites` object in the local storage. In the same spirit, its cookies are also removed from the `cookies` object in the local storage.

4 EVALUATION

In this section we will present a preliminary evaluation of our GRISEO system. Specifically, we would like to understand how much latency our approach adds to regular web browsing.

Page Rendering Time. Being implemented as an extension, GRISEO naturally adds an extra processing step to the critical path; a step which may increase page rendering times and degrade the overall user experience. To measure the added latency we proceeded as follows:

- We collected the top-100 most popular web sites in Europe according to Alexa.
- We visited the home web page of these websites (i) in regular browsing mode, (ii) in Incognito browsing mode, and (iii) using GRISEO extension.
- We measured the latency that our extension introduced.
- We visited each web site 50 times and took the median of these accesses.

To automate the rendering and measurements, we used the `selenium` python library. We understand that using an automated tool such as selenium may result in slightly different download times from those an ordinary user might experience. For example, when loading a long page (such as a site with a long list of news items), an ordinary user might glance the top few news items and then move to another web page even before the rest of the page downloads completely. On the contrary, selenium will wait to download the entire page with all the news items. Although we do understand that selenium might have a different browsing behaviour than ordinary users, this behaviour is the same for all three browsing modes that we test: regular, incognito and GRISEO. Thus, for the purpose of our experiments, which is to compare three browsing modes, selenium is satisfactory because in all three browsing modes behaves in the same way: it waits to download the entire page in all three modes.

Table 1 shows that average page rendering time. We see that the average page rendering time in regular browsing mode for the top-100 European Alexa sites is around 5.21 seconds. We see that incognito is a bit faster - around 5.24 seconds. Finally, we see that using GRISEO the average web page rendering time is close to 5.52 seconds. This amounts to about 5.9% of additional latency compared to Chrome regular mode.

Latency Overhead. Although GRISEO adds a little overhead, we would like to find out where this overhead is being spent. To minimize the network latency effects, we chose a nearby server and repeatedly accessed it 100 times. We then measured where is the time spent in the extension of GRISEO. Table 2 shows how the latency of GRISEO is distributed among the different modules of the extension. We see that 60% of the time is spent handling cookies: 43.80% is spent loading cookies, and 20.78% is spent saving them. Note, that our experiments are with white-listed web sites, and thus cookies have to be loaded and stored. The rest one third of the time left, is spent on Whitelist lookup and various tab handling events. At this point we can safely conclude that GRISEO is fast enough. We could try to optimize it a bit, esp. the Whitelist lookup time, but note that two thirds of its time is spent loading/storing cookies, a task which is difficult to optimize even further.

Table 1: Page rendering time for the different browsing modes. We visited the home pages of the top-100Alexa web sites. We see that the average time to render a page in regular mode is 5.21 seconds, in Incognito mode it is 5.24 seconds, and in GRISEO it is 5.52 seconds. This represents a mere 5.9% overhead.

Mode	Web page rendering time
Chrome regular mode	5.21 seconds
Chrome Incognito mode	5.24 seconds
Chrome with GRISEO	5.52 seconds

Table 2: Distribution of the latency among the various modules of GRISEO. We see a bite less than half of the time (43.80%) is spent spent on loading cookies and about 20.78% is spent on saving them. The rest of the time is Whitelist Lookup and Tab Event Handlers.

Module	Percentage of overall latency per module
Loading of Cookies	43.80 %
Whitelist Lookup	29.15 %
Window Close Event Handler	20.78 %
Tab Event Handlers	6.27 %

5 RELATED WORK

The preservation of the user's privacy on the web has drawn a significant body of research recent years. Indeed, there are numerous approaches aiming either to obfuscate the data a user generates while browsing the web (Papadopoulos et al., 2013; Sweeney, 2002; Kornilakis et al., 2019) or cut potential user tracking attempts.

Examples of the latter include the use of ad blockers and cookie blockers, that have been extensively used for a very long time on the web. Extensions such as Adblock Plus (Vallade, 2008), Ghostery (Signanini and Shnir, 2014), and CIC-AB (Lashkari et al., 2017) have been popular with users, because they block advertisers and their associated tracking. To respond to this blocking, advertisers and trackers have started detecting such blockers and go as far as refusing to serve their content to the sites using ad blockers (Iqbal et al., 2017; Mughees et al., 2017; Zhao et al., 2017). Our approach is slightly different: we do not block cookies - we allow cookies. We allow cookies because users need them in order to log in in their accounts online. What we do differently, is that we just do not allow cookies to survive across browsing sessions. This implies that advertisers can not track users from one session to the next. One might reasonably argue that one could allow first-party cookies (to enable log in), but block third-party cookies (to disable tracking). Although this sounds reasonable, content providers hunt down ad blockers very aggressively to the point that they do not serve content if they notice any blockers. To enable users to browse *all* web sites, our approach just uses incognito mode which accepts all cookies and is not being hunt down by content providers.

To help users browse the Internet anonymously, anonymizing networks, including the popular Tor system, have been widely deployed (Dingledine et al., 2004; Haque et al., 2019; Hoang et al., 2018; Yu et al., 2016). Using a network of proxies which communicate via encrypted messages, tor, in particular, manages to effectively hide the IP address of the client in traditional client-server web accesses. Our approach is slightly different. Tor, and similar networks, have a much stronger adversary model where they would like to hide communication from powerfully adversaries. Such adversaries may include entire States which may have physical access to the Internet Infrastructure, may influence their ISPs, and may enforce censorship. Our operational model is much simpler: we would like to help users browse the web continuously in incognito mode, and provide a seamless and safe transition to normal browsing mode when they want to log in to their accounts in the web sites they use. We would like to do this without increased tracking (that the normal browsing mode would enforce) and without blocking from the content providers (that an ad-blocker would suffer). The benefit of our approach is simple: "seamless web access with less tracking".

Chrome's Incognito mode, and similar "private" browsing modes have been under investigation to see "how much" privacy they actually provide and if they leak any information (Marrington et al., 2012; Wu et al., 2018). Leaks were found, or at least leaks were found where users did not expect them to exist. In the same spirit, traces left by Incognito browsing mode on the local disk could jeopardise the privacy of the users. To remedy the situation, a new file-system-based approach to incognito browsing was proposed and evaluated (Xu et al., 2015). We view our approach as complementary to this line of work. Indeed, that work focuses on exposing the weaknesses of the incognito mode, and eventually, on fixing these weaknesses and strengthening the privacy guarantees of the incognito mode. Our approach builds *on top* of the incognito mode. The stronger the incognito mode becomes, the better the privacy guarantees that GRISEO can offer as well.

Chrome and similar browsers offer users the possibility to create several browsing "profiles". Using this functionality, different users (probably members of a family or co-workers) can share the same browser: each user has a different profile. It is also possible for a single user to create several profiles: one for work, one for home, one for test, etc.. Using such profiles can provide privacy, but it is much more cumbersome that GRISEO. Indeed, using Chrome profiles, the user has to switch manually between different profiles, and has ensure not to make any mistake and browse web sites using the wrong profile. It seems that using profiles to provide privacy is even more cumbersome than using the incognito browsing mode.

To improve user privacy, Firefox has introduced multi-account containers (Mozilla Firefox, 2020). Each container has a different set of cookies. Users can have one container for web sites related to work, another container for the web sites related to shopping, another container for web sites related to travel, etc.Firefox, has also introduced temporary (i.e., disposable) containers (stoically, 2020). These containers provide compartmentalization of cookies: each container has its own cookie-store and can not share cookies with any other container. It seems that containers can be used to provide privacy and, to some extent, work similar to our approach. However, we think that there are significant differences between Firefox containers and GRISEO:

- Containers may become difficult to manage. Once the user starts creating several containers (home, work, travel, shopping, etc.), then the task of assigning web sites to containers may become cumbersome, if not error-prone. We think that the simple black-and-white approach of GRISEO is much easier to understand and much less prone to introduce user mistakes.
- The categorization of sites running in containers is based on *functionality*: work, home, shopping,

fun, etc.. On the contrary, the categorization of sites in GRISEO is based on *trust*: GRISEO asks users a simple question: *do you trust this site enough so as to white-list it?*

- In Firefox, web sites are assigned to containers and do not change unless the user takes some action. On the contrary, in GRISEO, white-listed web sites may automatically temporarily become black-listed as shown in figure 2, when included in a broader "umbrella" of a black-listed web site.

To summarize, we believe that our approach is orthogonal and complementary to previous work, as it builds on top of the two existing browsing modes that most browsers provide: incognito and regular.

6 CONCLUSION

Over the past few years browsers have started to provide a "privacy-friendly" browsing mode, called Incognito or Private mode. Although this mode is very useful, users many times frequently find themselves having the need to switch from Incognito to regular mode and vice versa, in order to log in into their accounts in several web sites (such as banks, email providers, social networks, etc.).

We believe that frequently switching back and forth between regular and Incognito browsing mode is tedious and error-prone. For this reason we have developed GRISEO: an approach that mixes the advantages of both worlds. It enables users to preserve their privacy by browsing in Incognito mode but also maintain the user experience of a regular mode when they want to visit a website that they trust, which requires from them to log in.

Our preliminary results show that GRISEO can be implemented in 600 lines of code. In addition, we see that the imposed overhead is practically negligible since for the average web page access, the rendering time while browsing in GRISEO mode is just 5.9% more than while in Chrome's Regular browsing mode. We believe that approaches like GRISEO, which (i) reduce tracking, (ii) are simple to use, and (iii) do not affect functionality will be increasingly useful in the near future.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the EU H2020-SU-ICT-03-2018 Project No. 830929 CyberSec4Europe (cybersec4europe.eu). The paper reflects only the authors' view and the Agency and the Commission are not responsible for any use that may be made of the information it contains.

REFERENCES

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

Agarwal, P., Joglekar, S., Papadopoulos, P., Sastry, N., and Kourtellis, N. (2020). Stop tracking me bro! differential tracking of user demographics on hyper-partisan websites.

Bashir, M. A. and Wilson, C. (2018). Diffusion of user tracking data in the online advertising ecosystem. *Proceedings on Privacy Enhancing Technologies*, 2018(4):85–103.

Cao, Y., Li, S., Wijmans, E., et al. (2017). (cross-) browser fingerprinting via os and hardware level features. In *Proceedings of Network and Distributed System Security Symposium*, NDSS'17.

Dingledine, R., Mathewson, N., and Syverson, P. (2004). Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC.

Google Developers (2020). Introduction to service worker. https://developers.google.com/web/ilt/pwa/introduction-to-service-worker.

Haque, A., Nisa, S. T., Bari, M., Anika, A. N., et al. (2019). Anonymity network tor and performance analysis of aranea; an iot based privacy-preserving router. *arXiv preprint arXiv:1906.01276*.

Hoang, N. P., Kintis, P., Antonakakis, M., and Polychronakis, M. (2018). An empirical study of the i2p anonymity network and its censorship resistance. In *Proceedings of the Internet Measurement Conference 2018*, pages 379–392.

Iqbal, U., Shafiq, Z., and Qian, Z. (2017). The ad wars: retrospective measurement and analysis of anti-adblock filter lists. In *Proceedings of the 2017 Internet Measurement Conference*, pages 171–183.

Klein, A. and Pinkas, B. (2019). Dns cache-based user tracking. In *Proceedings of Network and Distributed System Security Symposium*, NDSS'19.

Kornilakis, A., Papadopoulos, P., and Markatos, E. (2019). Incognitus: Privacy-preserving user interests in online social networks. In Fournaris, A. P., Lampropoulos, K., and Marín Tordera, E., editors, *Information and Operational Technology Security Systems*.

Lashkari, A. H., Seo, A., Gil, G. D., and Ghorbani, A. (2017). Cic-ab: Online ad blocker for browsers. In *2017 International Carnahan Conference on Security Technology (ICCST)*, pages 1–7. IEEE.

Marrington, A., Baggili, I., Al Ismail, T., and Al Kaf, A. (2012). Portable web browser forensics: A forensic examination of the privacy benefits of portable web browsers. In *2012 International Conference on Computer Systems and Industrial Informatics*, pages 1–6. IEEE.

Mozilla Firefox (2020). Firefox multi-account containers. https://addons.mozilla.org/en-GB/firefox/addon/multi-account-containers/.

Mughees, M. H., Qian, Z., and Shafiq, Z. (2017). Detecting anti ad-blockers in the wild. *Proceedings on Privacy Enhancing Technologies*, 2017(3):130–146.

Papadopoulos, E. P., Diamantaris, M., Papadopoulos, P., Petsas, T., Ioannidis, S., and Markatos, E. P. (2017a). The long-standing privacy debate: Mobile websites vs mobile apps. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 153–162, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

Papadopoulos, P., Kourtellis, N., and Markatos, E. (2019). Cookie synchronization: Everything you always wanted to know but were afraid to ask. In *The World Wide Web Conference*, pages 1432–1442.

Papadopoulos, P., Kourtellis, N., and Markatos, E. P. (2018a). The cost of digital advertisement: Comparing user and advertiser views. In *Proceedings of the 2018 World Wide Web Conference*, WWW'18, pages 1479–1489.

Papadopoulos, P., Kourtellis, N., and Markatos, E. P. (2018b). Exclusive: How the (synced) cookie monster breached my encrypted vpn session. In *Proceedings of the 11th European Workshop on Systems Security*, EuroSec'18, New York, NY, USA. Association for Computing Machinery.

Papadopoulos, P., Kourtellis, N., Rodriguez, P. R., and Laoutaris, N. (2017b). If you are not paying for it, you are the product: How much do advertisers pay to reach you? In *Proceedings of the 2017 Internet Measurement Conference*, pages 142–156.

Papadopoulos, P., Papadogiannakis, A., Polychronakis, M., Zarras, A., Holz, T., and Markatos, E. P. (2013). K-subscription: Privacy-preserving microblogging browsing through obfuscation. In *Proceedings of the 29th Annual Computer Security Applications Conference*, ACSAC '13.

Roesner, F., Kohno, T., and Wetherall, D. (2012). Detecting and defending against third-party tracking on the web. In *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pages 155–168.

Signanini, J. and Shnir, F. (2014). Ghostery. https://addons.mozilla.org/fr/android/addon/ghostery/.

stoically (2020). Temporary containers. https://addons.mozilla.org/en-GB/firefox/addon/temporary-containers/.

Sweeney, L. (2002). k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570.

Vallade, J. (2008). Adblock plus and the legal implications of online commercial-skipping. *Rutgers L. Rev.*, 61:823.

Wu, Y., Gupta, P., Wei, M., Acar, Y., Fahl, S., and Ur, B. (2018). Your secrets are safe: How browsers' explanations impact misconceptions about private browsing mode. In *Proceedings of the 2018 World Wide Web Conference*, pages 217–226.

Xu, M., Jang, Y., Xing, X., Kim, T., and Lee, W. (2015). Ucognito: Private browsing without tears. In *Proceedings of the 22nd acm sigsac conference on computer and communications security*, pages 438–449.

Yu, H., Lee, E., and Lee, S.-B. (2016). Symbiosis: Anti-censorship and anonymous web-browsing ecosystem. *IEEE Access*, 4:3547–3556.

Zhao, S., Wang, C., Kalra, A., Vaks, L., Borcea, C., and Chen, Y. (2017). Ad blocking and counter-ad blocking: Analysis of online ad blocker usage.