Bachelor thesis
Computer Science

Radboud University

Cache Cookies: searching for hidden browser storage

Author:
Patrick Verleg
s3049701

First supervisor/assessor:
Prof. dr. M.C.J.D van Eekelen
m.vakeekelen@cs.ru.nl

Second assessor:
Dr. ir. H.P.E. Vranken
Harald.Vranken@ou.nl

June 26, 2014

Abstract

Various ways are known to persistently store information in a browser to achieve cookie-like behaviour. These methods are collectively known as 'super cookies'. This research looks into various super cookie methods. Attention is given to the means a user has to regulate data storage and what is allowed by the cookiewet.

The main goal of this research is to determine whether a browser's cache can be used to store and retrieve a user's unique identification. The results show that this is the case, and these 'cache cookies' use a method so fundamentally entangled with the cache that there is no easy way to prevent them.

Because of cache cookies and other methods, the only way to prevent all tracking methods is to disable cache, history, cookies and plug-in storage.

Contents

Chapter 1

Introduction

The internet used to be a place where web site content was provided almost exclusively by the web site owner. Over the last decade, this has changed rapidly. Web sites are no longer isolated sites on the Internet, but form a tangled web where third parties provide social integration, advertisements, web analytics and more. It's not uncommon for websites to embed a dozen of third parties [6] and that number has grown over the last years [5].

As the web evolved, so did the techniques deployed by third parties to accurately track users on different pages and between different web sites. Roughly six business models can be distinguished that depend on embedding third party content [6]:

- **Advertising companies** combine browsing data to create elaborate user profiles used to serve ads.
- **Analytical services** aim to give the web site owner insight in visitor behaviour. Most analytic tools depend on JavaScript informing the third party on visitor's actions.
- **Social integration** offer integration with social media sites. Examples include Facebook's like button and Twitter's tweet button. Even users who are not logged in are tracked by some social buttons [21].
- **Content providers** host content that is embedded in the web site, such as video, audio and news.
- **Frontend services** host Javascript libraries in order to provide content or speed up page loading.
- **Hosting platforms** such as Akamai help content providers with distribution of their content.

Furthermore, law enforcement and intelligence agencies might use web tracking to partially undo anonymization efforts [32]. For example, the United

States' National Security Agency has shown interest in tracking cookies for this purpose [26].

Although visitor tracking to some extent is possible in all business models mentioned above, they are of vital importance to advertising networks. For serving relevant ads, they depend on user profiling: the process of gathering information specific to each visitor in order to customize web site content [3].

These profiles contain more than the user's gender or socioeconomic status. A publicly available segmentation of advertising network Epic Marketplace offered an interesting insight in to what extent users are being profiled [34]. Segmentations included getting pregnant and fertility, menopause and repairing bad credit.

With upcoming techniques like real time bidding, advertisement networks bid on available ad slots while a web page is loading. The amount offered for an ad slot depends on the profile of that specific visitor. For example, a higher bid can be made if the advertising network decides that a user is a better match to the product being advertised. In other words, advertisement networks that have more information can build better profiles and place a better substantiated bid.

Since one of the keys to better profiling is collecting more data, advertising networks are looking for more ways to identify users. Three techniques can be distinguished [6].

- **Data storage** depends on recognizing a user by storing uniquely identifiable data on their computer and retrieving it afterwards.
- **Active fingerprinting** depends on requesting properties from the device that is used to visit the web page. These properties include installed fonts, plug ins and supported MIME-types. Also fingerprinting based on a device's accelerometer has been shown to work [28].
- **Passive fingerprinting** is a fingerprinting method where input consists of properties that need not to be requested but are always provided by the visitor. These properties include IP-address, operating system and user agent. Since these properties are always sent, there is no way for a user to know whether he is fingerprinted.

In this research we will focus on ways to store data on the computer of a web site visitor. This is the most reliable technique, since the data stored is chosen by the web server and therefore uniqueness can be guaranteed. In other words: it is not possible to mix-up users. Fingerprinting heuristics lack this property, but they have shown remarkable accuracy in both test and real world environments. For example, heuristics on web browsers using both Flash and Java could uniquely identify of 94% the test cases and and changes in fingerprint data (such as a changing IP address) can be matched to the old fingerprint in 99% of the cases [2]. Similar methods have also been

tested on Hotmail and Bing, confirming their privacy risks: 80% of the hosts could be identified by their user agent and IP address alone, yielding to a similar effectiveness as obtained with cookie usage [11]. With at least 145 of the Internet's top 10,000 sites using fingerprinting, it is more widespread than previously thought [1].

In this research, we will try to answer our main research question:

> What methods are known to persistently store identifiable information in a browser and can cache contents be used to introduce a new way of persistently storing such information?

To answer this question, we will first consider the conventional way of storing information: cookies (chapter 2). Since HTTP is a stateless protocol [29], cookies were invented as a way to maintain browser state between consecutive visits. Storing data on a computer with cookies are therefore the most common way to store data on a visitor's computer and thus track visitors. Users, however, demand control over their online privacy. Studies on US citizens have shown that 65% of respondents found the idea of targeted advertising invasive and 54% said they did not like the idea of ad targeting [7][30]. Because tracking is such a controversial topic, we will be focussing on means the user has to protect his or her privacy throughout this research. Both technical means and legal aspects will be discussed.

We will continue our research to persistent storage with non-standard storage methods. These non-standard methods are known as super cookies (chapter 3). We will look into different types of super cookies and their properties. Furthermore, we will investigate how people can regulate super cookie placement and whether European regulation is also applicable to super cookies.

After investigating cookies and super cookies, we will present the main contribution of this research: a way to persistently store information in the browser's cache that can be used to uniquely identify a user (chapter 4). We will show that cache cookies provide a reliable way of identifying visitors in an advertising network scenario. Cache cookies' strong and weak points will be compared with other super cookie methods and performance and consequences will be discussed. We have implemented cache cookies in a proof of concept that works on all major browsers. The code for this proof of concept can be found in appendix A.

Chapter 2

Cookies for maintaining browser state

When browsing the web, all information sent and received by the browser uses the HTTP protocol [29]. Since HTTP is a stateless protocol, an addition is needed to maintain browser state (such as the user account logged in with). Lou Montulli, working with what would later be known as Netscape Communications, added support for retaining browser state by letting web sites store small pieces of text on the visitor's computer. These became known as cookies and were added to every outgoing HTTP request.

2.1 Storing and retrieving cookies

Both HTTP request and response messages may contain headers. Request headers can, amongst others, include preferred language, user agent and accepted encoding. Response headers typically contain properties like caching directives, content type and language.

When the server needs to place a cookie, it sets the `Set-Cookie` header with the cookie's content. For example, Wikipedia places a cookie with your estimated physical location.

```
Set-Cookie: GeoIP=NL:Nijmegen:51.8333:5.8667:v4;
            Domain=.wikipedia.org
```

The `Domain` keyword specifies for which web location the cookie will be attached to the request. Placement of cookies can not be guaranteed; client policy may dictate to ignore it. If the client saved the cookie, it will send the `Cookie` header on all subsequent requests in the request header.

```
Cookie: GeoIP=NL:Nijmegen:51.8333:5.8667:v4;
```

Cookies can also be stored and retrieved by the client with JavaScript.

```
document.cookie="GeoIP=NL:Nijmegen:51.8333:5.8667:v4;
                 Domain=.wikipedia.org";
```

At first sight, the possibility to store and retrieve cookies on client side seem to greatly extend possibilities. This is only partially true. Storing and retrieving cookies can easily be emulated by initializing HTTP requests with JavaScript. Saving cookies would be the same as making a request with the preferred cookie data as request argument, with a web server using that argument in the `Set-Cookie` header of the response and thus placing it in the browser. Reading cookies would be the same as requesting a page which returns the cookie data that was send in the request header. Furthermore, when using JavaScript, the actual cookie storage is just as uncertain since it also depends on the cookie policy of the client.

However, the same expression power can only be achieved if the server implements this kind of emulation. In practise, saving and reading cookies with JavaScript opens up new risks and possibilities. These risks involve cookie theft, leading to session hijacking. If third party's JavaScript is included in a page, the direct access to cookies enables the attacker to transfer cookies to his own domain.

2.1.1 Same-origin policy

Which cookies can be stored and retrieved for each domain is dictated by the same-origin policy. This web design principle states that information stored in the browser by one domain may later only be read or modified by the same domain. By implementing such a policy, web sites can peacefully coexist on a browser without interfering with each other. The same-origin policy dates back to version 2 of the Netscape browser [31].

The same-origin policy effectively isolates cookies, making them only accessible to the web site storing them. The `Domain` keyword we saw in the Wikipedia cookie thus only affects whether the cookie data is shared with its sub domains; it can never be shared with other domains. The same principle holds when storing and retrieving cookies with JavaScript.

Although the same-origin policy has been around for some time, no browser fully implements it for all browsing information. The failure to do so is one of the reasons for web data leakage [4]. All browsers do, however, enforce a same-origin policy for cookie data.

2.1.2 Third party cookies

One would suspect that tracking by a third party is not possible if a same-origin policy is enforced, because it blocks transferring identifiers to a third party. This was indeed the case for older cookie standards, which specified that sharing of cookies between servers should be limited [23][24], but newer standards relaxed this requirement [12]. Although cookies can still only

be stored and retrieved on their own domain, embedded external elements such as images or iframes allow for cookie interaction on their (third party) domain. The cookies that are set during the retrieval of external elements are called third party cookies. When cookies are used to track a user, they are frequently dubbed 'tracking cookies'.

So how do third party cookies work? The web site owner embeds a piece of external content on his web site in order display advertisements or enable social networks or web analtyics. When a page is loaded, so are the third party elements. The cookies set by these elements can only be read by their own domain, but since the same third parties are included on numerous web sites, each site contributes to the profile of the visitor. This profile is extended with information that is available to the third party, like the address of the web page it was embedded on.

Two common techniques make third party cookie tracking even more effective. Firstly, some web page properties, such as page title, can be requested by the third party's iframe because their properties are shared. These properties should therefore not contain identifiable information. Studies have shown that more than half of the web sites investigated directly leaked private information, such as names and addresses and even more leaked properties like user IDs [10][6]. Secondly, some third parties require embedding scripts directly into the first party's web site. The third party then becomes a first party, enabling techniques like cookie synchronisation between the first and the third party. Third party cookie policy can be evaded because the third party acts as a first party.

2.2 Protection against cookies

Both first and third party cookies raise privacy concerns. We will therefore look into ways to regulate cookie flow by considering the technical means available and the protection the Dutch law provides.

2.2.1 Technical means

All major desktop browsers[1] provide options to regulate cookie placement, like blocking all cookies or cookies originating from third parties. Options are available to remove cookies when the bowser is closed. Furthermore, all browsers allow fine-tuning the cookie policy on a per-site basis. Browser extensions are available to simplify this process. Mobile browsers[2] have significantly less options available and lack plug-in capabilities.

On Safari, third party cookies are blocked by default [16]. A third party cookie is not blocked when at least one cookie on the target domain has

[1] Safari, Chrome, Firefox and Internet Explorer.
[2] Chrome for Android, Safari for iOS.

been stored before, thus effectively blocking third party cookies only when the third party has never been visited as a first party. Mozilla thought about blocking third party cookies by default, but postponed implementation because further research was needed [27]. The definition of 'blocking' differs from browser to browser. For example, Internet Explorer only blocks the storage of a third party cookie (while reading is allowed), whereas Firefox blocks reading the cookie (while allowing storage). An ideal third party blocking system would disallow both reading and writing [4].

2.2.2 Regulation

The first attempt to regulate cookie placement was made by the European Union in the 2002 ePrivacy directive [19]. This directive mandates that member states implement laws that force web site owners to give users an opt-out possibility for cookie placement. Strictly necessary cookies that are requested by the user were excluded from this opt-out (article 5.3 of [19]). Because most member states did not enforce compliance the directive largely missed its goal [6].

The 2009 amendment to the ePrivacy directive replaced the opt-out with an explicit opt-in [20]. The directive was translated into Dutch law in the Telecommunicatiewet (article 11.7a) in 2013 and is commonly referred to as the 'cookie law'. The directive allowed to widen the scope of the member states' implementation and the Dutch government did so on two aspects. Firstly, the telecommunicatiewet requires explicit consent of the visitor, in contrast to the implicit consent required in other countries like the United Kingdom. Secondly, the burden of proof for collecting personal data is reversed. The latter means that the supervising organisation (Autoriteit Consument en Markt, formerly OPTA) does not have to prove that certain cookies are used for the collection of personal data. It is up to the other party to prove the contrary.

While privacy concerns regarding tracking cookies were the primary reason for the privacy directive, its scope is not limited to third party cookies. All cookies deemed not strictly necessary for a service require explicit permission. Examples of strictly necessary cookies include shopping cart information and language preferences, but in general do not include cookies set by advertising networks and web analytic companies. The minister stated that the classification of strictly necessary cookies is often a grey area, where necessity should be judged on a per-case basis [25].

In practise the majority of web sites do not obey the cookie law, because they either do not ask for consent or ask implicitly. An explicit consent requires to ask for permission before cookies are set, so a notification while browsing the web site will not suffice. Web site owners have to choose between potentially driving away users and risking a penalty from the supervisor. Many choose the latter, strengthened by the knowledge that Autoriteit

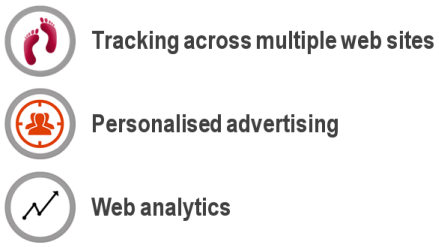

Figure 2.1: A mockup of standardised icons

Consument en Markt runs short on resources for investigation.

Suggestion for improvements

The cookie law has outspoken supporters and opponents. Three arguments can be put forward as of why the current law is not effective in protecting privacy of Dutch users.

Firstly, nearly every web site on the Internet uses cookies which are not strictly necessary. Therefore, they all have to ask for a users permission, leading to habituation. Users will consent without considering the risks and consequences.

Secondly, the law only recognizes two types of cookies: strictly necessary cookies and non-strictly necessary cookies. It can be argued that there is a huge difference between using cookies to track and profile users on the one hand, and using cookies to anonymously perform web site analytics on the other hand. Since the legislator did not make any difference, the same consent request is shown no matter how invasive the technique.

Thirdly, because the cookie law is focussed on informing users rather than protecting them, it leaves the possibility for web site owners to refuse access to users that do not consent to cookie placement. Although focussing on informing users is a legitimate political choice, it resulted in a very weak position for the web site's visitor.

A few suggestions can be made to improve the effectiveness of the current implementation of the law. To inform users more effectively, a consent form can be considered with standardized icons can be considered. These icons will clarify cookie consequences, just like the KijkWijzer or PEGI icons do for movies (figure 2.1).

Using these icons, web sites that use the most privacy invasive technologies are clearly recognised as such. This will likely help users to inform themselves. However, if the goal is to not only inform about cookies but also protect online privacy, more invasive legislation will have to be proposed. Requiring web sites to provide an opt-out that degrades user experience as little as possible will be a good first step. Denying access to users will then

belong to the past.

The cookie law has outspoken supporters and opponents and will likely be subject of debate for some time to come. The first change has already been proposed: a planned modification of the telecommunicatiewet will allow web analytic services without requiring consent [38].

Chapter 3

Super cookies

While the standard technique to track users is to use HTTP cookies, it is by no means the only way to do this. In the fast-developing Internet landscape, other ways have emerged to persistently store data and thus enable cookie-like behaviour. Some of them were invented to overcome cookie limitations, others materialized as a by-product of other features. These non-conventional ways of storing browser state are often collectively referred to as super cookies.

Most super cookies carry no more harm than cookies potentially do: they encode data for persistent storage. They differ in their storage space availability, whether data is transmitted in all communication or available on request, and whether their data is shared between browsers. But in a tracking scenario, they essentially do the same thing as traditional cookies, i.e. store a unique string so that a user can be recognised. The vast majority of Internet users is unaware of the different techniques used by super cookies and unaware of countermeasures that could be taken to protect online privacy. Advertising networks, whose business model depends on recognizing returning users, are well aware of this knowledge gap and we will see later on that they have deployed various types of super cookies.

Super cookie techniques can be combined to re-spawn deleted cookies, making them harder to remove. These returning cookies are called zombie cookies, and this questionable technique has been used by advertising companies [14], including major players like Microsoft [33].

While all super cookies techniques have the power to store data persistently in a browser, we can divide them in three major categories based on their characteristics.

3.1 Plug-in cookies

Probably one of the most well known types of super cookies originate from browser plug-ins. These plug-ins have their own methods available for per-

sistently storing data, while still being able to access HTTP cookies.

3.1.1 Adobe Flash

The Flash plug-in is the most used browser plug-in on the market, its publisher claims over 99% of the desktops have it installed [13]. Many computer vendors pre-install Flash on their devices and the Chrome browser even ships with a build-in version of Adobe's plug-in.

Using Flash, web site owners can use Local Shared Objects (LSOs) to store up to 100KB of data, which is stored within the user's operating system profile. Using profile-based storage instead of browser-based storage gives LSOs a major advantage over cookies from the perspective of a third party since it enables third parties to track users when they switch browsers. Until 2010, the same LSOs storage was used when browsing in private mode[1], ignoring its intended effect. The usage of LSOs has also been seen with major advertising networks, as well as their deployment to create zombie cookies [8]. LSO usages decreased somewhat after their usage drew negative attention from the media [15].

Protection

Over the last few years, Adobe made improvements when it comes to privacy. By supporting the `ClearSiteData` API since version 10.3, LSOs are now removed when HTTP cookies are cleared. Before version 10.3 was released in 2012, users had to go to Adobe's web site to clear LSOs which, as might be expected, the majority of the users never did. Alternatively, users could remove LSOs from the file system directly, but since LSOs are stored in hidden folders that option was not very popular either. Nowadays, Flash cookies still fall behind in terms of transparency compared to HTTP cookies though, since there is no easy way to see which LSOs are currently stored on a computer. Furthermore, fine-tuning of LSO permissions on a per-site basis is not possible via the browser interface.

The usage of LSOs can be blocked altogether with the Flash Settings Manager from version 10.3 onwards. The method of accessing the Settings Manager depends on the operating system, but it can usually be found in the system settings or control panel. Disabling LSOs will prevent tracking using Flash, but it might also break some Flash applications.

3.1.2 Microsoft Silverlight

Microsoft introduced the Silverlight plug-in in 2007 as a competitor to Adobe's Flash. Persistent storage for Silverlight cookies can be obtained

[1]Private mode (sometimes called InPrivate mode or incognito mode) is a special browser mode implemented by all browsers. History, cookies and other data is removed upon leaving private mode.

by using Isolated Storage. It can contain 100KB per site and is saved in the user's profile, just like LSOs. Isolated Storage is disabled in private mode. Microsoft does not release Silverlight penetration statistics but the penetration is estimated to be well over 50%, mainly due to automatic Windows Update installations.

Microsoft announced the retirement of their latest version of Silverlight in 2021 and future developments will probably be focussed on HTML 5. But until it is phased out, Silverlight provides the same type of storage as Flash does.

Protection

The efforts made by Adobe to increase privacy controls were not made by Microsoft. Although Silverlight does support private mode, it does not implement the `ClearSiteData` API. The only way to remove isolated storage is to remove files from a hidden folder in the file system or right click an Silverlight application and manage storage options. The latter being impossible when Silverlight is just used for storage and not displayed on the screen. In the same menu options for disabling isolated storage on all web sites can be found.

3.1.3 Java and other plug-ins

Java applets also provide mechanisms to persistently store data and interact with HTTP cookies. Because starting the Java Virtual Machine takes some time and the user gets notified that Java is running, Java is not an ideal candidate when it comes to super cookies. Its usage has not been observed by a third party web site.

Other plug-ins exist (e.g. Google Gears, Adobe Air), but are either not widely used or are deprecated (or both) and will therefore not be discussed.

3.1.4 Conclusion

Protection against plug-in cookies is hard. Plug-in cookies for Flash and Silverlight can be disabled, but it can be safely assumed that the vast majority of the users has no idea how to do that. Furthermore, using private mode will protect online privacy, but highly impractical to use as default mode.

With the usage of special browser extensions, plug-in cookies can be automatically deleted when closing the browser. The popular Ghostery[2] extensions provides this option, although it is disabled by default.

[2] `https://www.ghostery.com/`

3.2 HTML 5

Native client applications once held an advantage over the web when it comes easily access various data structures in persistent storage. Cookies were available but could not contain more than 4KB of data. Rich web applications were in need for more storage, which led to the proposal of HTML 5 storage techniques.

Early versions of the HTML 5 draft introduced a storage mechanisms known as global storage. If the draft were to be followed, it would have been possible to store data for arbitrary domains using JavaScript [37].

```
globalStorage["domain.example"].key = "value";
```

Data could even be stored so that any web site could access it.

```
globalStorage[""].key = "value";
```

All browser vendors refused to implement global storage due to its major violation of the same-origin policy. Later versions of the HTML 5 draft removed global storage and introduced two new storage types: local storage and session storage [39]. They both have a storage capacity of 5MB to store key-value pairs and they both comply with same-origin policy. They differ only in persistence and scope: session storage is deleted when the window is closed and only available in the current window, whereas local storage is kept after the window is closed and its contents can be shared between windows. Local storage is supported by all browsers and its properties allow for super cookie behaviour. As might be expected, its usage has already been observed [40].

Other HTML 5 storage mechanisms exist (e.g. database storage). These mechanisms operate under the same conditions as local storage does when it comes to scope, persistency and data capacity. Since they only differ in data structure, they pose the same threat to online privacy as local storage and will not be individually discussed.

Another technique worth mentioning is Internet Explorer's user data storage. This proprietary technique was introduced in Internet Explorer 5.5 and never made the transition to other browsers. It allows for storage up to 64KB in an XML format. It was declared obsolete with the introduction of Internet Explorer 7 but still works and can be combined with other techniques.

3.2.1 Protection

Browsers handle HTML 5 local storage exactly the same way as cookies when it comes to viewing, regulating and removing. Local storage will therefore be blocked when cookies are blocked (via the cookie policy, as described in

chapter 2). If a user insists on blocking only HTML 5 local storage, it can be achieved by blocking JavaScript. Extensions like NoScript[3] or Ghostery can help to define which scripts to block.

3.2.2 Conclusion

Local storage is well integrated in the browser. Since local storage is cleared when cookie data is emptied and can be viewed and fine-tuned on a per site basis, it does not pose and extra privacy threat over normal cookie usage.

3.3 Misusing browser features

One of the most interesting kinds of persistent storage techniques is storage using features which were never intended to use for storage.

3.3.1 Etags

Etags (entity tags) are part of the HTTP specification and provide a mechanism for cache validation [29]. Their usage is optional and supported by all major browsers. For every resource sent, the server adds a string which is used to validate the cache on subsequent requests. A collision-resistant hash function is often used due to its ability to create unique and reproducible etags. The server places the etag in the `ETag` response header of the HTTP request.

```
ETag: "17757e02095ed78048af2cf5030b09e8"
```

The client sends this value when requesting the resource in consecutive requests using the `If-None-Match` header.

```
If-None-Match: "17757e02095ed78048af2cf5030b09e8"
```

A `304 Not Modified` status code is sent if the etag matches the resource on the server. Otherwise the client's cache data is deemed to be stale and the new resource is sent.

The privacy problem resides in the fact that the method of etag generation cannot be known by the client. The client cannot verify whether the is really used for cache validator or is just a unique number used for tracking purposes. Since the the browser reveals the potentially unique etag on each request, it can be used to enables cookie-like behaviour. These etags are known as etags cookies an a number of web sites use them [9].

[3] `http://noscript.net/`

Protection

Etag headers can be stripped away using proxies that allow for header filtering, like Privoxy[4] or Proxomitron[5]. However, since header filtering does not work on encrypted connections this solution is not very solid (third parties could just switch to the HTTPS protocol to evade filtering). A browser extension could be made that disables last modified and etag storage, but no such extension seem to exist just yet.

3.3.2 Last modified

The last modified header provides another mechanism for cache validation. When enabled, the server adds a `Last Modified` header to every HTTP response.

```
Last-Modified: Mon, 9 Jun 2014 12:04:16 GMT
```

The browser adds this date on subsequent requests.

```
If-Modified-Since: Mon, 9 Jun 2014 12:04:16 GMT
```

The server checks whether the local resource is modified since the provided date and responds with either a `304 Not Modified` status code or the updated resource.

The format of these dates is provided in the HTTP specifications [29]. One would therefore expect that these dates do not provide enough entropy to allow for unique identifiers. However, in 2011 it was discovered that these date strings are not validated by the client and could thus contain any string [22]. This opens doors for the same usage as etag cookies, although usage has not been observed yet.

Protection

Last modified headers can be stripped away using a proxy or a browser extension, just like etags. No browser extension for last modified filtering seems to exist. A durable solution would be to bring down the last modifier entropy by enforcing strict date format and limiting the amount of information in the date string (e.g. remove seconds). Browsers have not implemented this yet.

3.3.3 Browser redirection

A special HTTP status code exists for resources that were permanently moved to another location. When a browser encounters such a status code it saves

[4] `http://www.privoxy.org/`
[5] `http://www.proxomitron.info/`

both the old and the new location, so that consecutive requests can be send directly to the new location. This property can be exploited to persistently store information [22]. A third party can redirect the browser to a page (for example: `domain.example/cookie`) that returns a redirection header to a unique location.

```
HTTP/1.1 301 Moved Permanently
Location: http://domain.example/cookie?id=5ed78048af2cf50
```

This redirection can take place in an iframe, in which case it is invisible to the user. When the third party requests `domain.example/cookie` after the redirection, the browser will load the new, unique location. The identifier is then known by both client and server and the user can be recognised. It has been reported that redirection is not always cached in all browsers, but it works in most configurations [18].

Protection

Protecting against tracking based on redirection is tricky. The header could be stripped by using a proxy or browser extension, but that would break parts of the Internet. Treating every permanent redirect as a non-permanent redirect would work because those redirects are not cached by the browser, but browser vendors would have to ignore the HTTP specification to do so. A plug-in could achieve the same result, but does not exist yet.

Header redirection will probably be noticed by the user if it cannot be hidden by using JavaScript. Therefore, blocking the required scripts will provide some protection.

3.3.4 Browser history

A super cookie technique that has been used by some major web sites was the CSS history leak. A third party would let the browser visit a few web pages from a collection of locations. The selection of these pages would encode a unique identification. Upon a later visits, the third party could generate all possible web pages from its collection and test whether the CSS `visited` pseudo class (frequently used to apply different colours to visited links) would change the layout of the elements. Since the CSS pseudo class revealed web pages the user visited, it allowed for both finger printing techniques and super cookie storage.

A few years ago, all browser vendors abandoned the CSS specification to protect their users' privacy. They made changes to the properties JavaScript had access to and took measures to prevent other layout-based attacks and some timing attacks [35]. Recently, new timing attacks were discovered by measuring the redraw events [36]. Just like with a lot of timing attacks, the

results will not always be accurate. Still, a very reliable assumption can be made. The new technique has not yet been observed in use by third parties.

Protection

As for the CSS history timing attacks, there are no third party means to prevent this kind of tracking except from blocking JavaScript. A durable solution can only be achieved if browser vendors implement countermeasures.

3.3.5 Techniques requiring user interaction

Some features can be misused to store data but require user interaction to do so. Most notable are the misuse of HTML 5 protocol and content handlers and HTTP authentication. Although these techniques can be very effective if the user is tricked into granting permission, they disqualify as silent super cookies and will not be thoroughly discussed.

3.3.6 Conclusion

Protection against techniques misusing browser features is very hard. Counter measures depend on the misused feature. Although all methods depending on cache can be theoretically countered by browser extensions, such extensions do not exist just yet. Protection of the history timing attack can only be achieved when countermeasures are implemented by browsers.

Until protection can be achieved, the only way to protect your privacy is to disable cache and history, or clear it frequently (this also happens when using private mode).

3.4 Regulation

In chapter 2 we discussed the impact of the cookie law on HTTP cookies. What does it say about super cookies?

The term 'cookie law' is somewhat misleading. The scope of the law is more extensive than just cookies. It includes anyone who wants to read or save data on a device. Flash storage, Silverlight storage and HTML 5 local storage fall within its scope. Even the placement of HTML, CSS and JavaScript has to be explicitly consented unless the files are strictly necessary for the requested service. In most cases that will be easy to prove. Since Flash and Silverlight storage is not strictly necessary when used for tracking, it will require explicit permission.

When we look at the misusage of browser features to store information, it becomes more complicated. A precaution is in place when discussing these

subjects. The Autoriteit Consument en Markt has not yet punished super cookie behaviour and until it does, this subject is grey area.

Let us consider etag header tracking. When placing an etag on a HTTP response, a explicit request to save the etag is given. The situation is comparable with cookies, which are also just a response header that requests persistent storage. Is an etag header strictly necessary? The answer to that will depend on how they are used. When used for tracking purposes, the answer is clearly 'no'. Because the burden of proof is reversed in the Dutch implementation, the web site owner has to prove that the unique string contained in an etag is not used for tracking. The same goes for tracking based on the last modified header.

When we look at redirect storage, the same reasoning applies. Since the 301 status code is an explicit request to store it permanently, explicit consent is clearly required.

Finally, let's consider storing data in a browser's history to be read with timing attacks later on. The minister addressed reading device properties in the Senate [25].

> "Reading device properties with the sole aim of providing the requested service falls under the exception of paragraph 3. Reading this information to track the device's user does not fall under this exception and does require explicit consent."

If 'device properties' were to be broadly interpreted, history reading will also require explicit consent.

The precise rules that apply to super cookies will be unknown until the Autoriteit Consument en Markt takes action and a case is brought to court. For now, it seems that the legislator was very well aware of super cookies and made their placement subject to the same conditions as cookies.

Nonetheless, from the perspective of a third party it is probably still smart to set multiple types of cookies. If the user consents once, it will be harder to remove all cookies when he changes his mind.

Chapter 4

Research: Cache Cookies

We have already seen super cookie techniques which depend on cached files, but they were merely used for their headers (etags, redirection and last modified). We will introduce a new way of persistently encoding data in the cache. In order to implement this new method, we have successfully circumvented browser's precautions for cache scanning. This results in a reliable super cookie technique that is nearly impossible to protect against without degrading browsing experience.

4.1 Method

Web resources can be cached to improve loading speed and reduce traffic. Caching can depend on client side policy, but in general the server's caching directions are strictly followed. The HTTP response header could suggest keeping cached items for many years. Items retrieved from cache can quickly be retrieved, and this property can be exploited.

To implement cache cookies, we will store a number of images in the cache. By checking which images are available, we can reconstruct the unique identification assigned to the user, even if all cookies have been deleted. Cache scanning behaviour will normally be prevented by browsers, but we will shortly see that we can easily circumvent this protection.

We will walk through the implemented method by showing a graphical overview and some code snippets. The full source code is available in appendix A.

4.1.1 Storing cache cookies

The client starts by checking if cached images are available. A special flag image will be cached by the client if cache cookies have previously been stored. The cache scanning function uses both the `complete` and the `width` and `height` properties to ensure full browser compatibility.

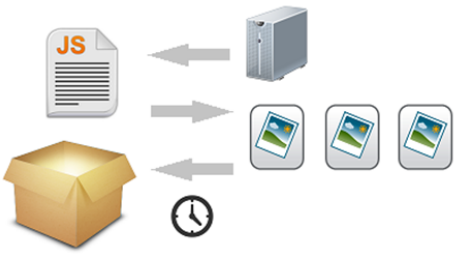

Figure 4.1: Cache cookies are stored in the browser's cache

```
function is_cached(img_url) {
    var img = document.createElement("img");
    img.src = img_url;
    return (img.width + img.height) > 0 || img.complete;
}

if (!(is_cached("/imgs/known.png"))) {...}
```

If the flag image is not available, cache cookies have not yet been stored on the device. The JavaScript code will request a unique identification from the server and translates that code into a set of images to load. There is a slight delay in the server's response, we will shortly see why. Figure 4.1 provides a graphically overview of the storage process.

4.1.2 Retrieving cache cookies

If the flag image discussed above is available, we expect images that encode the identifier to be cached as well. We scan for those images and retrieve the user identification.

```
var newAdId = 0;
for (var i = 0; i < MAX_IMAGES; i++) {
    if (is_cached("/imgs/" + i + ".png")) {
        newAdId += Math.pow(2, i);
    }
}
```

Browsers have incorporated precautions for cache scanning. If the `source` property is set on an element, the resource is directly requested and placed in the cache. This would normally be fatal for our super cookie since reading the cache once would result in all images being cached. These automatic requests cannot be cancelled by JavaScript. However, if we force a page reload all requests will be cancelled by the browser itself, preserving the integrity of our cache cookie. Since this reload is done in an iframe it will not be visible to the user.

Figure 4.2: Cache cookies are stored in the browser's cache

```
var y = MAX_IMAGES;
while (true) {
    if (y<0)
        break;

    if (adId-Math.pow(2, y) >= 0) {
        is_cached("/imgs/" + y + ".png");
        adId -= Math.pow(2, y);
    }
    y--;
}
```

One final precaution should be taken to avoid pollution of the cache. It should be guaranteed that a cache-scanned image cannot successfully load before a page reload is initiated. Since an advertisement network controls both the client side and the server side code, this can be implemented by adding a short delay in the server's response. Figure 4.2 provides an overview of the retrieval process.

4.2 Validation

One could ask how many different images would be needed for unique identification. There are roughly 10 billion internet connected devices around today. Because some have multiple, separated user accounts and others have no browsing capabilities, we can only make a rough estimate, but thirty to forty bits should be enough to assign a globally unique pseudonymous identifier. In the code example above, a `MAX_IMAGES` value of 40 would suffice for large advertisement networks.

The time needed for scanning these images can be benchmarked. Figure 4.3 shows the time required on our test machine[1]. Random bits were stored during these tests. A few conclusions can be drawn. Firstly, cache cookies are not qualified to store large amounts of data. Secondly, when

[1] An average desktop machine using Firefox 30.

scanning for a unique identification (30 to 40 bits), the sever should delay the response at least 200 milliseconds for reliable results. Since this timing is known by the server and the client, measures could be taken to guarantee that the cache will never be polluted. In order to do so, the JavaScript in the browser of the client has to constantly measure time and force a refresh if the 200 milliseconds are reached. The identification then fails, but the cache integrity will be preserved. In more advanced implementations, the time-out period can also be negotiated between client and server. The last conclusion is that cache scanning should be evaded as much as possible. This can be done by using cache cookies only to re-spawn HTTP cookies. Scanning will then only occur just after the HTTP cookies have been removed.

We now know retrieval time for a cache cookie, but how long does it take to store one? The answer depends on many factors. A request has to be made for each image to load. On average, a 40 bit identification will result in 20 HTTP request. The content of the served files can be as little as a few bytes, but HTTP 1.1 adds overhead: on average 200 to 400 bytes are received and transmitted per file. While the resulting traffic (around 8KB) is almost negligible, the large number of requests does add some delay. This delay can almost be completely eliminated by implementing the SPDY protocol [17]. If the server enables SPDY, all requests will be transferred over a single connection and all HTTP headers will be compressed.

We can conclude that cache cookies can be very reliable by taking counter measures that guarantee that the cache integrity will be preserved. Retrieving cookies takes some time and should therefore be avoided as much as possible, while storing cookies is very fast when the server is configured correctly.

4.3 Comparison with other techniques

In chapter 3 we already saw that storage techniques that misused browser features were extremely hard to prevent. How can cache cookies be prevented? And what are its strong and weak points from a advertising networking perspective?

4.3.1 Strong points

Unlike header-based cache methods (etags, last modified headers and redirection headers), cache cookies cannot be prevented by stripping or modifying headers. Cache cookies are not only harder to prevent than some other techniques, they also outclass some techniques in terms of reliability. The timing attacks used in the CSS history attack are based on an estimate and can therefore be influenced by other factors, like other processes running on the computer. Cache cookies also depend on timing, but the time difference in scanning and retrieving files can be kept high by delaying the server's

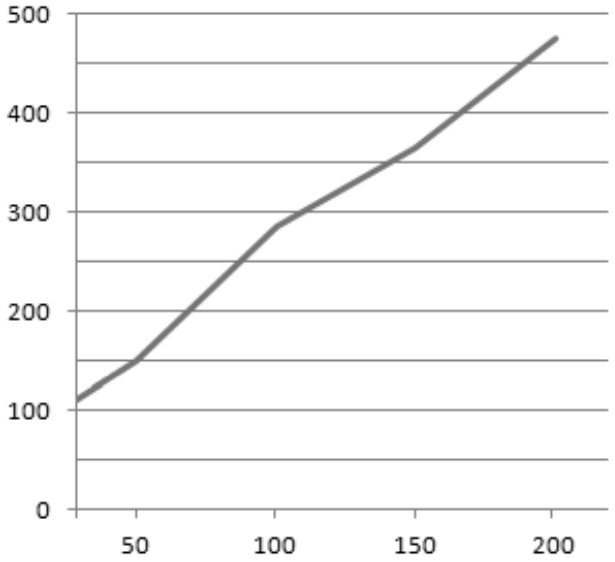

Figure 4.3: number of images scanned (vertical) vs. time in ms (horizontal)

response leading to a reliable identification. Since redirection seems to sometimes not be cached by the browser, cache cookies also outclass redirection cookies in terms of reliability [18].

We can conclude that cache cookies combine the reliability from header-based cookies while they are harder to prevent because they do not depend on headers that can be stripped or modified.

4.3.2 Weak points

Cache cookies can be prevented. Unlike storage based on etag and last modified headers, cache cookies will not work when JavaScript is disabled or blocked (with browser extensions like NoScript or Ghostery). Furthermore, just like all header-based cookies, they will fail if cache would be disabled (or frequently emptied), although that would leave a degraded browsing experience.

4.4 Reflection

Since presence of cache content is measured (rather than just a header), they touch the essence of caching. Finding a durable solution to prevent cache cookies is therefore not likely happen in the near future.

A strict same-origin policy for cache could be implemented to prevent cache cookies [4]. This is unlikely to happen though, not a single browser is around which strictly implements the same-origin policy for cache. Doing so would cripple some techniques frequently used nowadays, like multi

domain sign-in. Cache cookies would also fail if browsers would not cancel their requests when moving away from a web page. This is unlikely to be implemented because it would result in web pages continuing to load even after they have been left. In other words, cache cookies are here to stay.

If cache cookies are both reliable and hard to prevent, are they likely to be used? That depends. For a third party, a combination of super cookie techniques is probably the best guarantee to uniquely identify a user on subsequent visits. On the other hand, combining various *cache-based* super cookie methods offers few extra advantages, as it is safe to assume that the majority of the Internet users will not uses header modification via proxy or browser extensions.

The real addition of cache cookies probably twofold. Firstly, we have found a method that uses a technique so fundamentally entangled with the cache that no easy way of preventing seems to exist. Secondly, by adding another technique to the known methods, we have shown that persistent storage can take even more forms than previously known.

4.5 Related work

Non-destructive cache scanning has been proposed as a method to extract browsing history in 2011 [41]. The implementation of history scanning and the implementation of cache cookies share some properties but also differ on certain areas. They share iframe-reloading as their way to preserve cache integrity, but purposes and results vary between the two. Because the response speed of the third party cannot be controlled when scanning for external web pages, mixed results have been produced leading to a 84% to 90% success rate. These figures might seem high, but controlling the contacted web page is crucial for super cookies since a few incorrect cache hits could yield totally different identifiers.

Chapter 5

Conclusions

Many methods are available to persistently store information in a browser. Cookies are conventionally used for storage, but other techniques have emerged. These techniques known as super cookies, and different methods have different strong and weak points.

Cache could already be used to persistently store unique identification via super cookie methods based on headers. These methods could in theory be easily circumvented by altering headers via proxy or browser extensions. Due to the introduction of cache cookies, this is no longer the case. Cache cookies use a technique so fundamentally entangled with the cache that no easy solution seems to exist to prevent them.

Browsing the web without being tracked is hard because of cache cookies and other methods. The only way to prevent all tracking methods based on data storage is to disable cache, history, cookies and plug-in storage.

The cookiewet was introduced to inform users about all non-necessary cookies. Its scope is very properly defined: both cookies and super cookies seem to be affected by it. Because the cookie law only distinguishes two types of cookies, it is hard to make difference between different levels of privacy infringement. Standardised icons could be used to improve user's understanding. The law does not protect users from dubious tracking methods, if merely informs. New legislation has to be proposed if protection is required.

5.1 Further research

Not much research has been done with regard to super cookies. A thorough review of all browser aspects and their persistent storage probabilities can introduce even more ways of persistently storing information in a browser. This information should be kept in mind when drafting new web specifications, since browser vendors often have to balance between following the web specifications and protecting their users' privacy. This dilemma will be

evaded if privacy consequences were better thought of when creating web specifications.

Secondly, I would recommend to look into benchmarking the combination of various super cookie methods. Testing them on a large and representative web site should give insight in how often users clear their cookies and how many of those users can still be identified using super cookies.

It will also be interesting to create a browser aimed at preventing all known super cookie methods. We have already seen that some super cookie properties touch the core of browsing principles. Research will have to show what kind of browsing experience remains when some features are sacrificed in order to improve online privacy.

Peer-reviewed journals

[1] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. Fpdetective: Dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 1129–1140, New York, NY, USA, 2013. ACM.

[2] Peter Eckersley. How unique is your web browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, pages 1–18, Berlin, Heidelberg, 2010. Springer-Verlag.

[3] Magdalini Eirinaki and Michalis Vazirgiannis. Web mining for web personalization. *ACM Trans. Internet Technol.*, 3(1):1–27, February 2003.

[4] Collin Jackson, Andrew Bortz, Dan Boneh, and John C. Mitchell. Protecting browser state from web privacy attacks. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 737–744, New York, NY, USA, 2006. ACM.

[5] Balachander Krishnamurthy and Craig Wills. Privacy diffusion on the web: A longitudinal perspective. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 541–550, New York, NY, USA, 2009. ACM.

[6] Jonathan R. Mayer and John C. Mitchell. Third-party web tracking: Policy and technology. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 413–427, Washington, DC, USA, 2012. IEEE Computer Society.

[7] Aleecia M. McDonald and Lorrie Faith Cranor. Americans' attitudes about internet behavioral advertising practices. In *Proceedings of the 9th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '10, pages 63–72, New York, NY, USA, 2010. ACM.

Other references

[8] *Flash Cookies and Privacy.* Social Science Research Network Working Paper Series, 2010.

[9] *Flash Cookies and Privacy II: Now with HTML5 and ETag Respawning.* Social Science Research Network Working Paper Series, 2011.

[10] Privacy leakage vs. protection measures: the growing disconnect. In *Proceedings of the Web 2.0 Security & Privacy Workshop*, May 2011.

[11] Host fingerprinting and tracking on the web: Privacy and security implications. In *Proceedings of the 19th Annual Network & Distributed System Security Symposium.* NDSS, February 2012.

[12] U.C. Berkeley A. Barth. Rfc 6265: Http state management mechanism. `https://tools.ietf.org/html/rfc6265`, apr 2011.

[13] Adobe. Flash statistics: Pc penetration. `https://www.adobe.com/nl/products/flashplatformruntimes/statistics.html`, 2011.

[14] Lorrie Faith Cranor Aleecia M. McDonald. A survey of the use of adobe flash local shared objects to respawn http cookies. 2011.

[15] Lorrie Faith Cranor Aleecia M. McDonald. A survey of the use of adobe flash local shared objects to respawn http cookies, January 2011.

[16] Apple. Apple safari. `https://www.apple.com/safari/#privacy`, 2013.

[17] Peon Belshe. Spdy protocol. `https://mbelshe.github.io/SPDY-Specification/draft-mbelshe-spdy-00.xml`, feb 2012.

[18] Elie Bursztein. Tracking users that block cookies with a http redirect. `http://www.elie.im/blog/security/tracking-users-that-block-cookies-with-a-http-redirect`, jul 2011.

[19] European Commission. Directive 2002/58/ec of the european parliament and of the council of 12 july 2002. `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002L0058:en:HTML`, 2002.

[20] European Commission. Directive 2009/136/ec of the european parliament and of the council of 25 november 2009. http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32009L0136, 2009.

[21] Nik Cubrilovic. Logging out of facebook is not enough. http://nikcub.appspot.com/posts/logging-out-of-facebook-is-not-enough, sep 2011.

[22] Nik Cubrilovic. Persistent and unblockable cookies using http headers. https://www.nikcub.com/posts/persistent-and-unblockable-cookies-using-http-headers-2/, aug 2011.

[23] L. Montulli et al D. Kristol. Rfc 2109: Http state management mechanism. https://tools.ietf.org/html/rfc2109, feb 1997.

[24] L. Montulli et al D. Kristol. Rfc 2965: Http state management mechanism. https://tools.ietf.org/html/rfc2965, oct 2000.

[25] Eerste Kamer der Staten-Generaal. Wijziging van de telecommunicatiewet ter implementatie van de herziene telecommunicatierichtlijnen. https://zoek.officielebekendmakingen.nl/kst-32549-E.pdf, 2011.

[26] The Guardian. Tor stinks presentation. http://www.theguardian.com/world/interactive/2013/oct/04/tor-stinks-nsa-presentation-document, oct 2013.

[27] Jonathan Mayer. Web policy: a blog about technology, policy and law. http://webpolicy.org/2013/02/22/the-new-firefox-cookie-policy/, 2013.

[28] Nancy Owano. Accelerometer in phone has tracking potential, researchers find. http://phys.org/news/2013-10-accelerometer-tracking-potential.html, oct 2012.

[29] UC Irvine et al R. Fielding. Rfc 2616: Hypertext transfer protocol – http/1.1. http://www.w3.org/Protocols/rfc2616/rfc2616.html, jun 1999.

[30] TRUSTe Research. Privacy and online behavioral advertising. https://www.eff.org/files/TRUSTe-2011-Consumer-Behavioral-Advertising-Survey-Results.pdf, 2011.

[31] Jesse Ruderman. The same origin policy. http://www-archive.mozilla.org/projects/security/components/same-origin.html, 2001.

[32] Niklas Schmücker. Web tracking. 2011.

[33] Stanford Law School. Tracking the trackers: Microsoft advertising. http://cyberlaw.stanford.edu/blog/2011/08/tracking-trackers-microsoft-advertising, aug 2011.

[34] Stanford Law School. Tracking the trackers: To catch a history thief. http://cyberlaw.stanford.edu/blog/2011/07/tracking-trackers-catch-history-thief, jul 2011.

[35] Sid Stamm. Mozilla security blog: Plugging the css history leak. https://blog.mozilla.org/security/2010/03/31/plugging-the-css-history-leak/, mar 2010.

[36] Paul Stone. Pixel perfect timing attacks with html5. 2013.

[37] Alberto Trivero. Abusing html 5 structured client-side storage. http://www.scribd.com/doc/4012693/Abusing-HTML-5-Structured-Client-side-Storage, 2008.

[38] Volskrant. Kamp gaat toch de cookiewet wijzigen. http://www.volkskrant.nl/vk/nl/2694/Tech-Media/article/detail/3401513/2013/02/28/Kamp-gaat-toch-de-cookiewet-wijzigen.dhtml, feb 2013.

[39] W3C. Html5: W3c candidate recommendation 29 april 2014. http://www.w3.org/TR/2014/CR-html5-20140429/, 2014.

[40] Wired. Lawsuit targets mobile advertiser over sneaky html5 pseudo-cookies. http://www.wired.com/2010/09/html5-safari-exploit/, 2010.

[41] M. Zalewski. Rapid history extraction through non-destructive cache timing. http://lcamtuf.coredump.cx/cachetime/, 2011.

Appendix A

Code

The JavaScript code is based on the jQuery[1] framework with jquery-cookie[2] for cookie handling and purl[3] for url parsing.

```
function is_cached(img_url) {
    var imgEle = document.createElement("img");
    imgEle.src = img_url;
    return imgEle.complete || (imgEle.width+imgEle.height) > 0;
}

$(document).ready(function() {
    var MAX_IMAGES = 40;
    var origin = $.url().param("origin");
    var adId = $.cookie("adId");

    console.log("Adid: " + adId);

    if (adId) {
        console.log("Cookie available. adId is " + adId);
    }
    else {
        console.log("I do not know your adId yet");

        if (is_cached("/imgs/known.png")) {
            console.log("Some images are cached.");
            var start = new Date().getTime();

            // Load the MAX_IMAGES images. If all images are
            // gathered, reload the page.
```

[1] https://jquery.com/
[2] https://github.com/carhartl/jquery-cookie
[3] https://github.com/allmarkedup/purl

```
            var newAdId = 0;
            for (var i = 0; i < MAX_IMAGES; i++) {
                if (is_cached("/imgs/" + i + ".png")) {
                    console.log("Image " + i + " was cached.");
                    newAdId += Math.pow(2, i);
                }

                // Timing measures to guarantee cache integrity is
                // introduced here. In an advanced setup, timing
                // negotiation between client and server can be
                // implemented.
                if (end-start) > 200) {
                    $.cookie('adId', 'failed - scanning to slow');
                    location.reload();
                }
            }

            console.log("Cache constructed AdId was " + newAdId);

            // Set AdId cookie and reload before caching finishes.
            $.cookie("adId", newAdId);
            location.reload();
        }
        else {
            console.log("Cache was emptied or has not been filled yet");

            // Do normal ajax call to get new AdId
            // and load images for that adId.
            $.get("/getId/", function (adId) {
                var y = MAX_IMAGES;
                while (true) {
                    if (y<0)
                        break;

                    if (adId-Math.pow(2, y) >= 0) {
                        is_cached("/imgs/" + y + ".png");
                        adId -= Math.pow(2, y);
                    }
                    y--;
                }
            });
        }
    }
});
```