Rotten Cellar: Security and Privacy of the Browser Cache Revisited

Florian Dehling(✉), Tobias Mengel, and Luigi Lo Iacono

TH Köln University of Applied Sciences, Cologne, Germany
{florian.dehling,tobias.mengel,luigi.lo_iacono}@th-koeln.de

Abstract. Web browsers use HTTP caches to reduce the amount of data to be transferred over the network and allow Web pages to load faster. Content such as scripts, images, and style sheets, which are static most of the time or shared across multiple websites, are stored and loaded locally when recurring requests ask for cached resources. This behaviour can be exploited if the cache is based on a naive implementation. This paper summarises possible attacks on the browser cache and shows through extensive experiments that even modern web browsers still do not provide enough safeguards to protect their users. Moreover, the available built-in as well as addable cache controls offer rather limited functionality in terms of protection and ease of use. Due to the volatile and inhomogeneous APIs for controlling the cache in modern browsers, the development of enhanced user-centric cache controls remains—until further notice—in the hands of browser manufacturers.

Keywords: Browser cache · Security · Privacy

1 Introduction

Large distributed systems such as the web require technologies that provide high scalability. A mechanism that enables to reduce the amount of data to be transferred between the client and the origin server drastically is the caching of resources. Besides web caching systems, such as proxy caches and content distribution networks (CDN), the browser-internal cache plays an important role in reducing the amount of data transmitted over the network. This is done by storing static content temporarily on the user's system and loading it as required. The consequence of reusing a file at a later time is the question of its validity and freshness. Even though the browser cache has not yet achieved public fame in terms of security and privacy risks, several attack strategies exist (e.g. [10,16,26]) that emphasise the urgency to look at this topic more thoroughly.

This paper contributes towards this claim by capturing the current usage of HTTP browser caches, summarising security and privacy issues, as well as pointing out the necessity for future research on this topic. For this purpose, experiments are performed to explore the influence of browser caches on quality of

A. Askarov et al. (Eds.): NordSec 2019, LNCS 11875, pp. 20–36, 2019.
https://doi.org/10.1007/978-3-030-35055-0_2

service metrics, as well as to analyse the content of the browser cache in relation to shared resources. Moreover, the browser cache's handling of TLS-protected resources is investigated by carrying out a repetition experiment. More specifically, it is examined how browser caches handle resources that are transferred with different TLS certificates for the same origin domain. Finally, approaches and tools are investigated that allow users to improve their security and privacy when using the browser cache.

The remainder of this paper is structured as follows. First, the foundations of HTTP caching are laid with a particular focus on the browser cache. In Sect. 3, possible risks for security and privacy of users are summarised. In order to record the current use of the browser cache, Sect. 4 describes several experiments that were conducted and discusses the obtained results. Amongst them are the achievable data transfer savings as well as the files that are reused from the browser cache likewise by a large number of websites. Another experiment investigates the relationship between the validity of TLS certificates and the behaviour of the browser cache. Possibilities for users to influence the behaviour of the browser cache are discussed in Sect. 5. Beside comparing the procedures to completely delete the browser cache of well-known web browsers, available browser add-ons for extended control of the cache are examined. Last but not least, Sect. 6 discusses the results and presents some suggestions that could contribute to improving a user's security and privacy when browsing the web while maintaining the benefits of browser caching.

2 HTTP Caching Background

Transmitting data in large distributed systems can be expensive, especially if the bandwidth is limited somehow. As modern websites consists of numerous files, technologies are used that reduce the amount of data to be transmitted. The strategy considered here stores static components of a website on the clients system, in order to be able to call them up if necessary without renewed transmission. While so-called caches are used in numerous types on the internet, this paper is limited to the HTTP browser cache.

If a website contains static content like e.g. images or scripts, web browsers can store those files locally inside their respective browser cache. Once the client requests the corresponding website again, previously stored files can be loaded from the cache instead of requesting and transmitting them from the origin server again. This technology is part of the *Hypertext Transfer Protocol*, specified by RFC 2616 [12] and revised in RFC 7234 [11]. As not every resource of a website is suitable to be cached and even the content of supposedly static files may changes over time, a web server can send instructions on how the browser cache should behave and handle cached resources. This is done via the `Cache-Control` header in the corresponding HTTP response. In the following, the basic directives are listed and described briefly.

`no-cache`. Forces caches to deliver the request to the source server to check the validity of each request before releasing a cached copy.

`no-store`. The browser is strictly forbidden to temporarily store the response from the server. This statement is usually used if the response contains sensitive data.

`private/public`. Using the instruction `private`, responses can only be stored in the browser cache as they may contain private information. `Public` allows caching in shared caches, even if the response requires HTTP authentication.

`max-age`. This statement specifies the maximum time period in seconds beginning with the time of the initial request in which a stored response can be reused. If for example a response is assigned with the Cache-Control header directive `max-age=86400`, the corresponding resource can be reused up to one day.

Once a resource has been stored in the browser cache, the question arises as to whether this file is up-to-date. As specified in RFC 7234, two procedures can be used for this kind of verification and which enable a browser to perform so-called *explicit* caching. The *freshness lifetime* rates the validity of a resource by features which were previously defined by the web server. The HTTP header `Expires` or the `Cache-Control` header directive `max-age` can be used to define a period of time within which a resource may be reused by caches. A *freshness validation* describes a process to check the validity of a resource by consulting the web server but without transferring the actual payload. Using conditional requests [13], indicators are transmitted that provide information about the version of the cached resource. This can be done using the `ETag` header, which contains an opaque string describing the version of the resource, or by using time information such as the `Date` or `Last-Modified` header. One of these three features can be used by the client to send a conditional request to the server. If the latest version of the requested resource is present at the client's cache, the server responds with a status code `304 Not Modified`. If the resource invalidated in the meantime, it will be retransmitted in the response body.

Missing validation-features could theoretically cause a client to store a file for an indefinite time. To prevent this, web browsers perform so-called *implicit* caching if the server does not provide any features for *freshness lifetime* or *freshness validation*. The procedure of implicit caching depends on the implementation of the web browser and has been investigated in [22].

3 Security and Privacy Implications of the Browser Cache

In addition to the advantages that browser caches offer to users, they can also be a gateway to cyberattacks. These can be divided into two types, (1) those affecting the user's privacy and (2) those placing malicious code on the victim's computer. Several possible attack vectors have been detected and discussed in the literature already and are summarised in the following.

3.1 Browser Cache Poisoning

Browser Cache Poisoning attacks consists of placing manipulated files to the affected browser cache [15,19,26]. The procedure is as follows. The victim first visits a website which is described by an HTML document that also includes instructions to load further documents like JavaScript files. The attacker replaces one of those additional resources with his poisoned version. This can contain malicious code that gets executed if the document is interpreted by the victim's web browser. To infiltrate his modified file, the attacker can make use of a Man-In-The-Middle attack (MITM). This enables him to compromise the communication between the victim and the web server which is requested to deliver the content for the appropriated website. Beside manipulating the actual content, the attacker also ensures that the Cache-Control header enables the browser cache to store the file as long as possible without performing an validation process. If the victim visits the same website again at a later point of time, the web browser will load the manipulated file from the browser cache and executes the malicious code. At this time, the attacker does not need to have access to the communication between the victim and the requested web server as the malicious file was stored at the victims browser cache at the time of the actual attack. In order to achieve greater impact, the attacker will try to compromise files that are used by multiple websites likewise and thus requested by the victims browser more frequently. Files that are distributed via CDNs are a promising solution for this purpose. The risk of such an attack increases when resources are transmitted via unsecured connections.

3.2 User Tracking

Beside Browser Cache Poisoning attacks, the browser cache offers vulnerabilities that affect the user's privacy. This can be done in several ways. One approach is to place user-based-identifiers in HTML documents. In this case, a server-defined ID is assigned to e.g. a hidden DOM-element. If the appropriated HTML document is cached by the victims web browser, it will be loaded containing the identifier, when the victim is visiting the website at a later time. Using a script, the embedded ID can be extracted and transmitted to the server. On this way, both page visits can be associated and the user is identified [14].

Analysing loading times, also called Cache Timing Attacks (CTAs), is another possibility to track users online activities with the help of the browser cache [8]. This approach makes usage of one of the core features of this technology, the drastic shortening of loading times. In order to explore which websites already have been visited by the victim, a script can be used to request significant resources, like logos, of the websites to test. By measuring the time that elapses until the resource is loaded, an attacker can determine whether the victim has already visited a specific page or not. Assuming that websites cause the browser cache to store static content like logos, a short loading time occurs if the requested content was loaded from the browser cache, whereas a longer loading time is caused by loading the file from a web server.

This procedure is also possible without the use of runtime-analysing scripts [10]. To do so, a sequence of requests, two to the own origin and one to a resource of a website to be tested, needs to be defined in a HTML document in immediate succession. The first request to the own origin triggers the start of the time analysis. Next, a resource of a website of interest gets requested like in the procedure that has been described before. As now, there is no application logic in the client's browser that could measure the time elapsing while loading the external resource, a second request to the HTML document's origin gets triggered immediately after the external resource was loaded completely. By analysing the time difference between the incoming requests on the origin server, conclusions about the duration of the loading time for the external resource can be drawn on the server side. The practical implementation of this method requires the consideration of parallel loading of sub-resources. The attacker thus needs to ensure that all requests of a sequence are executed successively after the previous one is completed. This can be archived using the `onload` event of a DOM element. Even though JavaScript is required to implement this, the evaluation takes place on the server side, so no suspicious code is visible to the client.

Using both procedures, it is also possible to obtain geographic information of website visitors. By checking locally related pages it is possible, in particular for a limited number of users to be identified, to infer their location or to identify them on the basis of their location [16].

The third way to implement a tracking mechanism based on the browser cache is utilising HTTP headers that are originally intended to be used for validating cached resources. The HTTP headers `Last-Modified` and `ETag` are suitable to transport user-based identifiers. These are stored together with the corresponding resource to the browser cache. If a user is visiting a website that requires this previously stored resources, the web browser performs a *freshness validation* by transmitting the `Last-Modified` or the `ETag` header values. On this way, the user-based identifier is available on the server side and can be used in order to derive a usage behaviour [17].

4 Browser Cache Experiments

To examine the impact of browser caches with respect to the current state of the web, an extensive analysis is carried out. The top 500 URLs of Tranco [20] retrieved on August 19, 2019 serves as data basis. It results from a combination of well-known website ranking lists like Alexa, Cisco Umbrella, Majestic and Quantcast which has been prepared as a reliable and manipulation-free database for security and privacy research. Using a Chromium devtools extension, each URL of the list gets requested five times in a row with a time interval of 20 s. The recorded traffic is exported in the HTTP Archive (HAR) [23] format for further analysis. Since data which is stored persistently on the client's hard disk is of particular interest, the memory-cache, which is used by Chromium to temporary store content for opened tabs, is cleared before each page call. The records

thus can be reduced to the usage of the disk-cache. In case of examinations with activated browser cache, every website is called once before starting the measurements to fill the cache. Since the content and thus also the transmitted data of a website can be time-dependent, the measurement conditions described in the following are carried out in parallel.

4.1 Effects on Network Performance

The first part of the analysis focuses on the effect of a browser cache on the transmitted data volume when a website is accessed. For this purpose, the transferred data volume of incoming responses that do not originate form the disk-cache is computed. In order to avoid measurement errors, only the sub-resources of a website that are present in the data set of all measurement conditions are considered. The median of resulting data volume for the five available measurements is determined and used for further evaluation. This procedure is performed for three test cases:

(a) activated browser cache
(b) deactivated browser cache (via option in the devtools)
(c) browser cache limited to same-origin and a *https* pattern in the URL

Test case (c) was achieved by using a developed browser extension that prevents the browser from loading resources from the cache that do not originate from the host of the page the user is attempting to access. As the affiliation of a resource is determined by its URL, the usage of CDNs that do not include the full name of the host, specified for the main HTML document, leads to an exclusion from the browser cache. In addition, caching is limited to URLs starting with the pattern *https*.

The results show a median of saved data of 95.9% for the comparison of activated and deactivated browser cache (test cases (a) and (b)). As shown in Fig. 1, half of the requested websites reach a data saving of between 89.7% and 98.9% in this measuring condition. Restricting the use of the browser cache to resources from the originally requested origin and a *https* schema in the URL results in a significantly different outcome. As shown in Fig. 2, half of the requested websites archive a data saving of 0.35% to 77.7%. The median reaches a value of 37%.

The comparison of the data savings of the two evaluations clarifies the number of resources that originate from sources that are not initially related to the originally requested website. Some might originate from CDNs that are operated by the provider of the website, but whose URL cannot simply be traced back to the URL of the main HTML document. Another part are third-party resources which are not associated with the originally requested website.

The evaluation of measurement conditions (a) and (b), the comparison of activated and deactivated browser cache, clearly points out the importance of this technology with regard to the scalability of a distributed system like the web. The renouncement of a browser cache thus leads to a drastic increase of required bandwidth, which is unacceptable, especially with mobile applications.

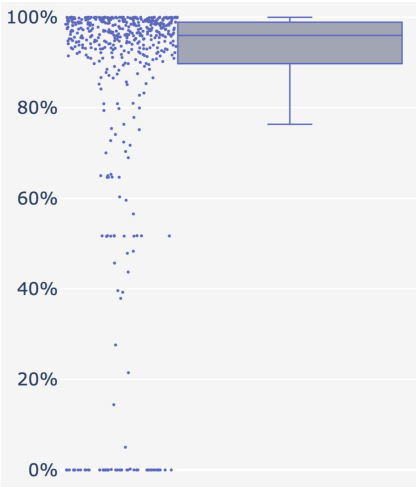

Fig. 1. Percentage of data saving by using the browser cache of Chromium for ubuntu. Median: 95.9%, q1: 89.7%, q3: 98.9%, min: 0%, max: 100%

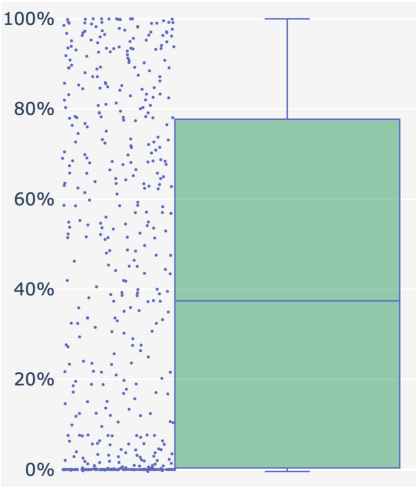

Fig. 2. Percentage of data saving by using the browser cache of Chromium for ubuntu with limitation to same-origin resources and *https* URL patterns. Median: 37%, q1: 0.35%, q2: 77.7%, min: 0%, max: 100%

4.2 Current Security and Privacy Risk Assessment

The second part of the analysis focuses on potential threats caused by the browser cache. Browser Cache Poisoning attacks are particularly dangerous if they are applied on resources that are used by several websites likewise. In order to quantify this danger, the records are analysed to find cached resources that are shared among several websites. We again use the top 500 websites by Tranco [20] and capture the HTTP traffic with activated browser cache. The logs are analysed as follows. If a sub-resource is loaded from the browser cache, the URL

Table 1. The 10 most referenced resources by the Tranco top 500 websites

URL of resource	Web page referencing resource
https://www.google-analytics.com/analytics.js	202 (40.4%)
https://connect.facebook.net/en_US/fbevents.js	111 (22.2%)
https://connect.facebook.net/signals/plugins/inferredEvents.js?v=2.9	111 (22.2%)
https://www.googletagservices.com/tag/js/gpt.js	81 (16.2%)
https://tpc.googlesyndication.com/safeframe/1-0-35/html/container.html	80 (16%)
https://securepubads.g.doubleclick.net/gpt/pubads_impl_2019082201.js	79 (15.8%)
https://www.googletagservices.com/activeview/js/current/osd.js?cb=%2Fr20100101	67 (13.4%)
https://www.googletagservices.com/activeview/js/current/osd_listener.js?cache=r20110914	65 (13%)
https://sb.scorecardresearch.com/beacon.js	64 (12.8%)
https://securepubads.g.doubleclick.net/gpt/pubads_impl_rendering_2019082201.js	60 (12%)

of the initiating website is captured. This leads to a list of web pages loading the same resource from the browser cache.

Table 1 contains the top 10 shared resources, sorted descending by the amount of requesting websites. 40% of the websites requested in this analysis reference the script *analytics.js* delivered by the host *google-analytics* which is located in the browser cache. Compromising this file in would therefore have impact on a large number of websites.

Reviewing Browser Cache Poisoning Attacks. To investigate the relevance of Browser Cache Poisoning attacks on current web browsers, a further experiment was carried out. This draws particular attention to the relationship between the validity of TLS certificates and resources loaded from the browser cache. A similar investigation has been done in [15] in 2015. In order to verify how modern web browsers deal with this kind of attack, a renewed assessment was done. The investigation is based on the assumption that files in the browser cache are assigned solely based on a single cache key i.e. the URL, but no distinction is made as to whether the resource was transferred via a valid TLS connection or not. A valid TLS connection is defined to be based on a certificate whose authenticity is fully and successfully verified by the web browser. The experiment set up (see Fig. 3) is described in the following.

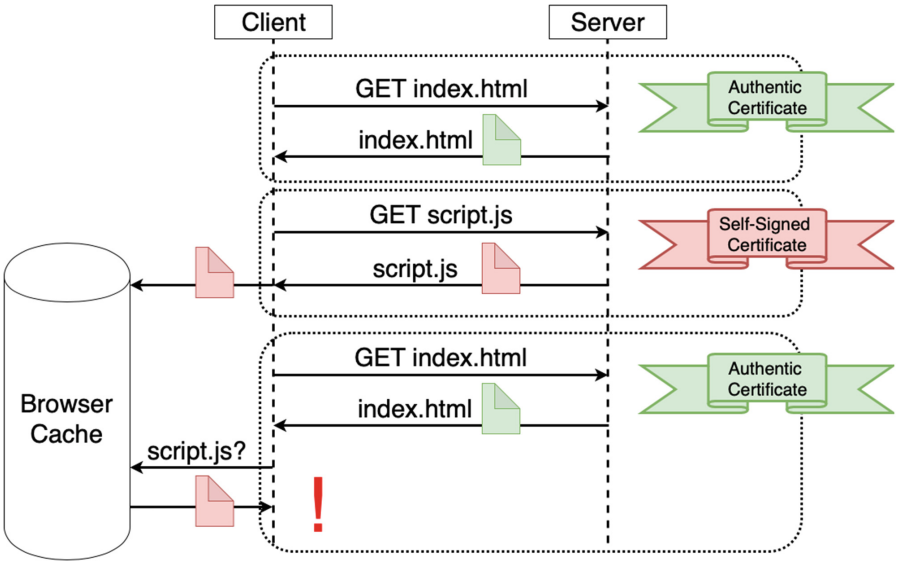

Fig. 3. Procedure of Browser Cache Poisoning attack performed in order to test the relationship between TLS certificates and cached resources

The client visits a website that is providing a valid TLS certificate. This website serves an index HTML file and a JavaScript file as sub-resource. Both

files are delivered with cache statements that forbid the browser to load them from the cache. In order to simulate a MITM attack, the delivering web server changes its configuration and uses a self-signed certificate that is not trusted by the browser. The client now reloads the website and receives an TLS warning message. After he successfully clicked through the warning, the web server delivers its data as usual. As before, two resources are transferred, an index HTML page and a JavaScript file. The latter now contains malicious code, but does not provide any caching instructions. After a further change of the configuration, back to the valid TLS certificate, the client updates the requested page again. Although he now uses a trusted TLS connection, the browser loads the manipulated JavaScript file from the browser cache. This will be the case until the user explicitly instructs his web browser to request all resources from the origin, or clears the cache.

To perform this experiment, an nginx web server is used to distribute the HTML and JavaScript file. A Let's Encrypt certificate was issued for the server, in addition a self-signed certificate was created. The server configuration is used to switch between the two certificates, whereby the self-signed certificate simulates a MITM attack. When changing the certificates, also the content of the JavaScript file changes, whose function consists of inserting the currently configured scenario (good certificate/bad certificate) as string into an object of the HTML page. The selected scenario assumes that the HTML file is not cached by the browser in both configurations. In the case of a valid certificate, the JavaScript file should not be cached, but the MITM attack should save it persistently. Figure 4 shows the Cache-Control header configuration defined on the server.

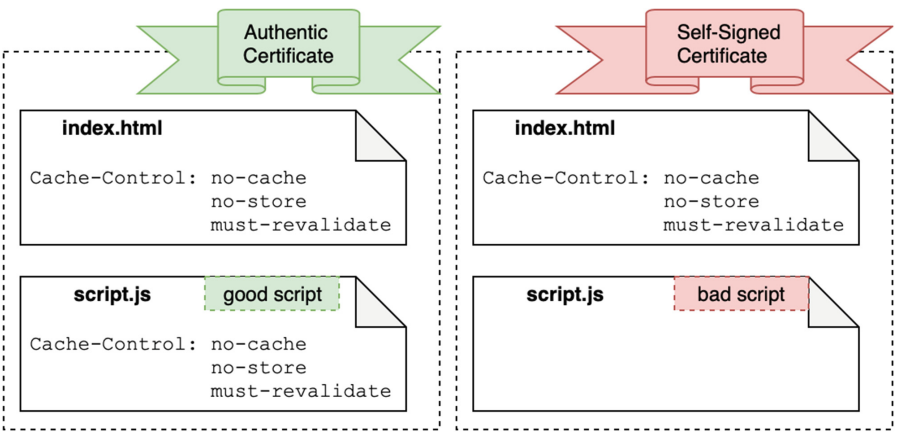

Fig. 4. Cache-Control header of the resources provided by the server during the simulated Browser Cache Poisoning attack

The results (see Table 2) show that all tested web browsers are vulnerable to the chosen attack. Due to the cache implementation of the Chrome browser, the

JavaScript file delivered with the valid certificate gets cached already, so that an explicit reload of the page need to be done to load the manipulated script. For all browsers, a TLS certificate warning needs to be clicked through before accessing the web site transmitted when using a self-signed TLS certificate. After switching back to the authentic certificate, all browsers load the manipulated script from the browser cache. Chrome, Internet Explorer and Edge also cache the HTML file, which results in the certificate warning in the URL-bar remaining for the time being. Firefox updates the TLS information in the address-bar; Safari does not distinguish between accepted self-signed and authentic certificates. After the web browsers are closed and restarted, Internet Explorer, Edge and Safari still load the manipulated JavaScript file from the browser cache, this time without any certificate warnings. The scenario constructed is probably rare in reality, since the described attack can be prevented by simple mechanisms such as Sub-resource Integrity [7]. Nevertheless, it shows that none of the browsers tested takes the authenticity of the certificate into account when caching.

Table 2. Impacts of the simulated browser cache poisoning attack

	Chrome v76 (Windows & MacOS)	Firefox v69 (Windows & MacOS)	Internet Explorer v11 (Windows)	Edge v44 (Windows)	Safari v12 (MacOS)
Caches malicious script (self-signed certificate)	(Yes) after forced page reload	Yes	Yes	Yes	Yes
Loads malicious script (valid certificate)	Yes[a]	Yes	Yes[a]	Yes[a]	Yes
Loads malicious script after browser restart (valid certificate)	No	No	Yes	Yes	Yes

[a]As the index.html file is loaded from the cache too, the certificate warning in the URL-bar is still present.

5 User-Centred Mitigation Strategies

Vulnerabilities of browser caches can affect the security and privacy of users. The analyses described in the previous section points out that modern web browsers still not provide sufficient mechanisms to prevent such attacks. The likewise usage of shared resources shown in Table 1 increases the impact of an incident. In order to improve the security of the browser cache without extensive changes in the implementation of web browsers, users could make use of additional tools. In the following, user-centred mitigation strategies will be presented and analysed. As it is well known that security-relevant technologies require a high degree of usability in order to achieve the intended improvements in practice [27], we reviewed the following mechanisms under consideration of efficiency and task adequacy. First, we will highlight the simplest method imaginable, deleting

the browser cache. Due to the different implementations of the web browsers, the usability of this function is not sufficient in all cases. Afterwards, available browser extensions are examined which influence the functionality of the browser cache to achieve a higher level of security and privacy.

5.1 How to Clear the Browser Cache

To evaluate the usability of the cache delete function for the web browsers Firefox, Chrome, Safari, Internet Explorer and Edge, the procedure is examined by focusing on the user's effort. Table 3 shows the number of clicks necessary to clear the cache.

The clear cache function of Firefox is located in the security and privacy area of the browser settings. Five clicks are required to use it.

Chrome allows to open the corresponding dialog without navigating to the general browser settings. The deletion of browser data can be limited to a certain period of time. As the last hour is selected as default, users first must adjust the period to the total interval of time. If this is done once, the cache can be cleared within four clicks.

The Safari browser does not offer any option to clear the cache when using the default configuration. With a total of five clicks, the developer settings must be activated first to enable the option to empty the cache memory. If this is done, two clicks are necessary.

To clear the browser cache in Internet Explorer, the appropriated dialog can be accessed using the *Security* sub-menu of the browser's main-menu. Unfortunately it is marked with the term *browser history*. Four clicks are required to clear the cache.

Edge requires a procedure similar to Firefox. The corresponding function can be found in the security and privacy section located in general settings. Five clicks need to be done to clear the cache.

With the exception of Firefox and Edge, the procedures for deleting the browser cache differ. Especially the necessity of activating the developer options on Safari seems unnecessarily complicated. Chrome, Firefox, Internet Explorer and Edge also offer a shortcut to display the dialog box for deleting browser data. In this case, Firefox will open a window which is different from the one accessible via the menu. Similar to Chrome a time period must be selected for which the data should be deleted.

Table 3. Number of clicks required to clear the browser cache

	Firefox	Chrome	Safari	Internet Explorer	Edge
Number of clicks	5	4	5 + 2	4	5

Instead of clearing the browser cache, a private browsing session can be used for all considered web browsers. The resources cached during private sessions get cleared once all private browsing windows are closed.

5.2 Tool-Based Solutions

Frequently clearing the full browser cache does not seem to be an efficient way to deal with the present security and privacy threats when considering the amount of data saved using this technology. A way of defining cache policies for sensitive websites, such as online banking or risky resources like the ones originating from different origins, would be more desirable. Browser add-ons can provide additional functionality that could lead to an improvement for the user's security and privacy. We analysed extensions that are available at Google Chrome Web Store, Firefox Add-ons and GitHub considering a functionality which exceeds clearing the browser cache as well as a usable design which enables ordinary users to make use of them. We used the keyword "cache" to search for browser plugins at Chrome and Firefox extension stores as well as "browser cache extension" to find relevant projects on GitHub. Due to a large number of results, the analysis is restricted to the first 60 hits.

Extensions found at the Chrome App Store can be grouped into three categories. The largest group (26 results) provides only deletion of the cache or in some cases further storages such as the browsing history. Another category (8 results) refers to applications that offer to display cached elements or to render web pages using resources stored in the browser cache. The third category (3 results) contains applications that can manipulate the caching behaviour. These include *NoCache* [24], *PowerCache* [18] and *Supercache* [21]. *NoCache* provides a button to disable the browser cache. Using *PowerCache* it can be individually specified which elements should not be loaded from the browser cache. This is done by using a regex pattern to filter the requested URLs. Requests that are manipulated by the extension are listed in the add-on. This application is primarily intended for developers and thus is not designed to be usable for ordinary web users. *Super Cache* is using a simple and minimalist user interface. One can specify whether stylesheets, images or scripts should be loaded from the cache. The options *Default* (no manipulation), *Cache* or *NoStore* can be set. These settings can be configured for each website individually. As this extension only manipulates the Cache-Control header directive of response headers, it can only prevent resources from being stored to the browser cache. If a file is already located there, it will be used to render the website anyway.

The analysis of Firefox browser extensions was done in the same way. Among the first 60 results, 13 applications can be found which can clear the cache, six applications that can display cache contents and two applications that allow deactivation of the cache (*Toggle Cache* [25] and *Cache It Out* [6]). Five applications provide reloading of a page without using files from the browser cache. As this concept seems to be similar to clearing the cache or performing a full reload of a website, these extensions are neglected. Due to a restriction of the implementation of the extension API used by Firefox, it is not possible to control the behaviour of the browser cache in detail. This would require the manipulation of a HTTP request with regard to the caching behaviour before it is processed by the browser, which is not possible in Firefox extensions.

The search on GitHub revealed eleven applications that allow to clear the browser cache. Four projects offer the possibility to display content from the cache. Only one application (Opera-disable-cache [9]) can deactivate the browser cache. As it is implemented for the Opera browser, it will not be discussed further. Table 4 provides an overview of the features of the remaining five applications that can control the caching behaviour.

Table 4. Comparison of five browser extensions that allow to control caching behaviour

	NoCache (Chrome)	PowerCache (Chrome)	SuperCache (Chrome)	Toggle Cache (Firefox)	Cache It Out (Firefox)
GUI	X	✓	✓	X	X
Page specific settings	X	X	✓	X	X
Resource-type specific settings	X	✓	✓	X	X
View cached resources	X	✓	X	X	X
Multiple settings per website	X	X	✓	X	X

(✓: feature provided, X: feature not provided)

6 Discussion

The security and privacy risks presented in this paper, as well as the results of the experiments carried out, demonstrate that modern web browsers are still providing insufficient precautions to protect users. The analysis of present user-centred mitigation strategies points out that further work needs to be done in order to compensate the shortcomings of current browser cache implementations. The following section should take up the most important issues and give recommendations on how these can be solved effectively.

Web Browser Extensions. In order to improve the browser cache with regard to security and privacy, it would be conceivable to develop browser extensions that control the behaviour of the cache according to users' needs. Using browser add-ons, shortcomings of the current caching procedure could be diminished without depending on extensive and time-consuming developments of improved web browsers. When looking at the APIs for Chrome and Firefox extensions, we noticed that there is no direct way to manipulate the behaviour of the browser cache [1,5]. By setting the Cache-Control header directive `no-cache` in the request header, it is possible to prevent Chrome from loading a resource from the cache. The documentation of Chrome contradicts this observation, but describes this information as unstable. [2]. It would be desirable if browser vendors would provide well-documented interfaces by which developers can create high-performance solutions that give users more control on the behaviour of their web browsers.

Tracking via Browser Cache. As discussed in Sect. 3, the browser cache can be utilised to attack users' privacy. By using individual identifiers, a web tracking mechanism can be implemented, very similar to the functionality of cookies. We have not yet been able to establish any more detailed investigations into this matter, but consider this to be absolutely necessary. While the introduction of the General Data Protection Regulation (GDPR) increased users' awareness on privacy related risks of cookies due to the omnipresence of cookie info boxes on many websites, the risks related to the browser cache are far less present and seem to have received little attention so far. We plead for a comprehensive analysis of possible further uses of the browser cache with regard to the risks for the privacy of users. This includes but is not limited to a critical consideration of *freshness validation* mechanisms as they can be misused to exchange user identifiers using `ETag` or `Last-Modified` header values.

Cache Keys and TLS Certificates. Restricting the cache key to the URL for resources transferred over TLS secured connections can lead to urgent security risks. However, this practice seems incomprehensible, since information about the authenticity of the certificate used in the TLS connection is available in the web browser. Resources transferred via TLS connections based on different certificates should not be considered identical. In fact, this contradicts the security service of authenticity. A resource should therefore be bound to the certificate used during the transfer. Bypassing the browser cache for unauthentic TLS connections could prevent Browser Cache Poisoning attacks by limiting the impact of MITM attacks to the moment of the actual attack. Investigations also revealed that the devtools of the Firefox web browser, unlike the information in the URL-bar, do not distinguish between authenticated and, for example, self-signed certificates. In both cases a green icon appears beside the transferred resources displayed in the network tab. This does not seem to be sufficiently differentiated and could lead to confusion among developers.

Browser Cache Partitioning. To prevent attacks that could harm the privacy of users, Chrome plans to partition the cache [4]. This is to be achieved by using the origin URL of the website requested by the user as an additional key for a cached resource. This approach is to be welcomed as it can prevent attacks like CTAs. Developers of Firefox are also considering this improvement [3]. It remains to be seen whether this gets actually implemented.

7 Conclusion

HTTP browser caches are important components of distributed systems like the web as they enable these systems to scale at large. We show that half of the 500 most popular websites from the Tranco list archive savings between 89% and 99% of the data volume to be transmitted when browser caching is enabled. Thus, disabling the browser cache has severe performance and economic consequences for service providers and end users respectively.

At the same time though, browser caches can be misused to perform user tracking and to persistently store malicious code on the client's computer. The current practice does not sufficiently balance security and performance trade-offs. By taking a closer look at the files stored inside the cache we found resources that are used by up to 40% of the examined websites. As these resources most commonly stem from third parties providing analytics or advertising services, the potential for user tracking is inherent. Moreover, we found that modern web browsers still do not distinguish between valid and self-signed TLS certificates when storing and reusing resources from the browser cache. Although this weakness opens the door for Browser Cache Poisoning attacks and has been known since 2015 [15], it is still exploitable in the most current versions of major browsers.

Hence, providing a more fine-grained control of the cache policy enforced by the browser is essential. Browsers themselves offer caching by default and simple settings to clear the cache. In some cases these settings are even well hidden from the end user. Some browser extensions aim at closing this gap with more enhanced features. However, our analysis of these extensions emphasises that they provide only very fragmented functionalities with most of them only going slightly beyond browsers' builtin functionalities.

Overall, we argue that more research is required in respect to browser caches and their relation to the security and privacy of end users. This would enable browser vendors to close existing vulnerabilities and provide more elaborated cache control settings for end users as well as APIs for developers and researchers. The latter will foster the iterations towards novel approaches to balance performance and security.

References

1. Chrome APIs - Google Chrome. https://developer.chrome.com/extensions/api_index. Accessed 05 Sept 2019
2. chrome.webRequest - Google Chrome. https://developer.chrome.com/extensions/webRequest. Accessed 05 Sept 2019
3. Double-keyed HTTP cache Issue #904 whatwg/fetch. https://github.com/whatwg/fetch/issues/904. Accessed 05 Sept 2019
4. Partition the HTTP Cache - Chrome Platform Status. https://www.chromestatus.com/feature/5730772021411840. Accessed Sept 05 2019
5. WebExtensions. https://developer.mozilla.org/de/docs/Mozilla/Add-ons/WebExtensions. Accessed 05 Sept 2019
6. Firefox user 13863091: Cache it out. https://addons.mozilla.org/en-US/firefox/addon/cache-it-out/. Accessed 05 Sept 2019
7. Akhawe, D., Braun, F., Marier, F., Weinberger, J.: Subresource Integrity. W3c Reccomendation, W3C (2016). https://www.w3.org/TR/SRI/. Accessed 05 Sept 2019
8. Bansal, C., Preibusch, S., Milic-Frayling, N.: Cache timing attacks revisited: efficient and repeatable browser history, OS and network sniffing. In: Federrath, H., Gollmann, D. (eds.) SEC 2015. IAICT, vol. 455, pp. 97–111. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18467-8_7

9. Digital, b.: Opera extension to disable browser cache, perfect for developers: biati-digital/opera-disable-cache. https://github.com/biati-digital/Opera-disable-cache. Accessed 05 Sept 2019
10. Felten, E.W., Schneider, M.A.: Timing attacks on web privacy. In: Proceedings of the 7th ACM Conference on Computer and Communications Security, CCS 2000, Athens, Greece, pp. 25–32. ACM, New York (2000). https://doi.org/10.1145/352600.352606
11. Fielding, M.N.R., Reschke, J.: RFC 7234: hypertext transfer protocol (HTTP/1.1): caching. Technical report RFC 7234, IETF (2014)
12. Fielding, R., et al.: RFC 2616: hypertext transfer protocol-(HTTP/1.1). Technical report RFC 2616, IETF (1999)
13. Fielding, R., Reschke, J.: RFC 7232: hypertext transfer protocol (HTTP/1.1): conditional requests. Technical report RFC 7232, IETF (2014)
14. Fleischer, G.: Implementing web tracking. In: Black Hat USA 2012 Conference Briefings, pp. 1–37 (2012)
15. Jia, Y., Chen, Y., Dong, X., Saxena, P., Mao, J., Liang, Z.: Man-in-the-browser-cache: persisting HTTPS attacks via browser cache poisoning. Comput. Secur. **55**, 62–80 (2015). https://doi.org/10.1016/j.cose.2015.07.004
16. Jia, Y., Dong, X., Liang, Z., Saxena, P.: I know where you've been: geo-inference attacks via the browser cache. IEEE Internet Comput. **19**(1), 44–53 (2015). https://doi.org/10.1109/MIC.2014.103
17. Juels, A., Jakobsson, M., Jagatic, T.N.: Cache cookies for browser authentication. In: 2006 IEEE Symposium on Security and Privacy (S P 2006), pp. 5–305, May 2006. https://doi.org/10.1109/SP.2006.8
18. Khachatryan, A.: Power cache. https://chrome.google.com/webstore/detail/power-cache/famkodflhompmapangljedfdcfeligih?hl=de. Accessed 05 Sept 2019
19. Kuppan, L.: Attacking with HTML5 (2010). https://media.blackhat.com/bh-ad-10/Kuppan/Blackhat-AD-2010-Kuppan-Attacking-with-HTML5-wp.pdf. Accessed 05 Sept 2019
20. Le Pochat, V., Van Goethem, T., Tajalizadehkhoob, S., Korczynski, M., Joosen, W.: Tranco: a research-oriented top sites ranking hardened against manipulation. In: Proceedings 2019 Network and Distributed System Security Symposium, San Diego, CA. Internet Society (2019). https://doi.org/10.14722/ndss.2019.23386
21. Mathur, T.: Super-cache. https://chrome.google.com/webstore/detail/super-cache/fglobbnbihckpkodmeefhagijjcjnbeh?hl=de. Accessed 05 Sept 2019
22. Nguyen, H.V., Lo Iacono, L., Federrath, H.: Systematic analysis of web browser caches. In: Proceedings of the 2nd International Conference on Web Studies, WS.2 2018, Paris, France, pp. 64–71. ACM, New York (2018). https://doi.org/10.1145/3240431.3240443
23. Odvarko, J., Jain, A., Davies, A.: HTTP Archive (HAR) format (2019). https://w3c.github.io/web-performance/specs/HAR/Overview.html Accessed 05 Sept 2019
24. Oluwaseye: No cache. https://chrome.google.com/webstore/detail/no-cache/hckocmggmdfdnjjomghmhllibmdobdll. Accessed 05 Sept 2019
25. Reimer, M.: Toggle cache. https://addons.mozilla.org/de/firefox/addon/togglecache/?src=search. Accessed 05 Sept 2019

26. Saltzman, R., Sharabani, A.: Active man in the middle attacks. OWASP AU (2009). http://www.security-science.com/pdf/active-man-in-the-middle.pdf. Accessed 05 Sept 2019
27. Whitten, A., Tygar, J.D.: Why Johnny can't encrypt: a usability evaluation of PGP 5.0. In: Proceedings of the 8th Conference on USENIX Security Symposium, vol. 8, SSYM 1999, Washington, D.C., pp. 14–14. USENIX Association, Berkeley (1999). http://dl.acm.org/citation.cfm?id=1251421.1251435