Security considerations around the usage of client-side storage APIs

Stefano Belloro (BBC)

Alexios Mylonas (Bournemouth University)

Technical Report No. BUCSR-2018-01

January 12 2018

ABSTRACT

Web Storage, Indexed Database API and Web SQL Database are primitives that allow web browsers to store information in the client in a much more advanced way compared to other techniques such as HTTP Cookies. They were originally introduced with the goal of enhancing the capabilities of websites, however, they are often exploited as a way of tracking users across multiple sessions and websites.

This work is divided in two parts. First, it quantifies the usage of these three primitives in the context of user tracking. This is done by performing a large-scale analysis on the usage of these techniques in the wild. The results highlight that code snippets belonging to those primitives can be found in tracking scripts at a surprising high rate, suggesting that user tracking is a major use case of these technologies.

The second part reviews of the effectiveness of the removal of client-side storage data in modern browsers. A web application, built for specifically for this study, is used to highlight that it is often extremely hard, if not impossible, for users to remove personal data stored using the three primitives considered. This finding has significant implications, because those techniques are often uses as vector for cookie resurrection.

CONTENTS

1. INTRODUCTION

The first webpage ever created was made up purely of hypertext. Nothing complex: no colours, no pictures, and no animations. Just simple text and, here and there, occasional references to other pages that the user can browse at will. One could argue that it was hypertext in its simplest form. Fast-forward to our time and the experience is very different. The web now looks nicer but, worryingly, it often seems to know a lot about its users. This is because, over the years, a few techniques to store information on the client have been developed.

This work will focus on three client-side persistence storage primitives: Web Storage, Indexed Database API and WebSQL Database and their usage in the context of user tracking.

Chapter 2 provides an historical and technical **contextualisation** of those three client-side storage mechanisms. This section will mainly focus on the characteristics of the primitives considered, studying the documents provided by the two principal bodies responsible for the definition of the specifications: the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG). A brief overview of other similar techniques will also be given, starting from the very first mechanism for client-side data storage, HTTP cookies. The chapter will also give an historical overview of the circumstances that led to the specification of those primitives.

Chapter 3 lists a selection of **relevant studies** from the literature. They focus on three main themes: **confidentially and integrity** of the data, user **tracking** and **preventive measures** against it. While Chapter 2 describes the features of the primitives and mechanisms for client-side data storage, this chapter focus on studies and analysis on how those techniques have been used in the wild.

Chapter 4 presents an audit on the usage of Web Storage, Indexed Database API and WebSQL Database in the wild and presents a way of quantifying the adoption of these storage mechanisms by user-tracking agents. To the best of the writer's knowledge, this is the first large-scale study that considers the usage of those three storage systems and quantifies the use case of user tracking. This section will show that adoption of these techniques by trackers is much higher that what previous studies have found and that tracking agents are indeed a major employer of those techniques.

Chapter 5 presents an analysis on the control that users have over data stored locally on their devices. In particular, this section will show that the task of removing personal data can be tricky, if not impossible, on some browsers. Moreover, it will show a few idiosyncrasies in the way certain browsers implement private browsing mode. These findings are important because they highlight that users of mobile devices are at higher risks of being tracked against their will.

Chapter 6 recapitulates the key findings of this work, especially in the context of similar works mentioned in Chapter 3. It also provides a few suggestions for further work.

2. BACKGROUND

HTTP is a stateless protocol (Fielding et al, 1999). This means that once a client-server transaction ends, the server will not retain session information or status about any of the clients involved. Nonetheless, the web is nowadays a very customised ecosystem. This is the result of an evolution that spanned through several years. The following paragraphs will discuss a series of techniques for persistent data storage in the client.

2.1. COOKIES

HTTP cookies are the first technique that was developed to allow browsers to link together a series of stateless HTTP requests with stateful information (Barth, 2011a). An HTTP cookie is a short piece of data that a website sends to a visiting client, either via HTTP response headers or by using client-side scripting. The client is then expected to save this information and send it back to the server when making subsequent HTTP requests. HTTP cookies are a technology that was proposed, and later patented, by Montulli (1995) while he was working at Netscape Communications. He borrowed the term 'cookie', and part of the logic behind them, from Unix's 'magic cookies' (McIlroy and Kernighan, 1979). The information stored in a HTTP cookie is in the form of a string, which can be used by the server to identify a certain session id, certain user preferences, or a certain user.

While the idea behind HTTP cookies was not entirely novel, the quick adoption of this technology by browser vendors revolutionised the web. It allowed websites to easily support features that entailed advanced customisation, such as virtual shopping carts and token-based user authentication. Each cookie is associated to an origin, which is a combination of the hostname, the port number and the protocol used by the web application (Barth, 2011b). This is based on a concept known as 'same-origin policy', which has been the cornerstone of browser security since the early days of the web (Shepherd, 2017). The same-origin policy regulates and restricts how a document or a script from one origin can interact with a resource loaded from another origin.

However, since a webpage can contain resources from multiple origins, HTTP cookies are often used to identify and track users, not only across different browsing sessions, but also across different websites. Over the years, both Internet users and legislators have become more aware of the privacy implications of third-party tracking (Kristol, 2001), also thanks to the interested generated by the popular press (Etzioni, 1999). Modern web browsers allow users to remove HTTP cookies or to disable them completely. Moreover, in some jurisdictions, web sites are requested to proactively ask for consent in order to save and process personal data.

It should to be noted that, despite being so impactful and ubiquitous, this technology has got its limitations. For example, one of the limits of storing data through HTTP cookies is in the amount of information that can actually be stored. Web browsers tend to limit the size of cookies to a few thousands characters. Contrary to what a few papers in the literature seem to imply, this limitation is not imposed by the specification written by Kristol and Montulli (2009). In fact, that document recommends that user agents should not impose fixed limits to the size and amounts of cookie they support.

Web browsers, however, not only limit the length of HTTP cookies, but they also apply constraints to their quantity, allowing only a few dozens of them per origin. Several studies can be found online providing an overall view of the limits that different vendors set to HTTP cookies (Manico, 2009; Roberts, 2013).

Limiting the size and the amount of HTTP cookies that a web page can set is a reasonable implementation decision, considering the performance implications that come with their usage. Indeed, cookies are not an ideal way of transferring large amounts of data, mainly because they are transmitted through HTTP requests.

Larger cookies, if they were at all possible, would bloat the size of the HTTP requests, resulting in slower client-server transactions. This would eventually mean slower webpages. Moreover, they would also be problematic for web servers. Constantin (2016) demonstrated that most web servers would not be able to cope with very large HTTP cookies. He estimated that the capacity of the average web server is limited to 3 cookies of size 4096 bytes.

As the web evolved, the desire for different and more capacious ways for storing structured data on the client started to emerge. Over the years, several client-based storage technologies appeared. Most of them, such as Local Shared object of Adobe Flash (Adobe Systems, 2012), Oracle Java (Oracle 2017), Microsoft Silverlight (Microsoft, 2017) and Google Gears (Google Code, 2008), were made available through browser plug-ins. Others, like Internet Explorer's UserData (Microsoft Developer Network, 2011) were vendor-specific technologies, only available on a single browser.

Towards the end of the noughties, the web community witnessed a great leap forward in terms of capabilities of front-end technologies (Anthes, 2012). Web developers were putting a lot of effort into making the web experience more 'app-like' and browser vendors were keen on supporting functionalities that could replace third-party plug-ins, such as Adobe Flash (Jobs, 2010). A new version of HTML was in the process of being drafted (Hickson and Hyatt, 2008) in a joint effort of the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG). Even though the work on the specification was still in progress, the new standard, often refereed to as HTML5, quickly obtained industry-wide recognition (Cox, 2011). In this context a few new proposals for persistent data storage in the client stared to take shape. These include Web Storage, Web SQL Database and Indexed Database API, which are described in the next subsections.

2.2. WEB STORAGE

Web storage (WHATWG, 2017) is a specification that allows web applications to create a persistent key-value store in the browser, the content of which is maintained either until the end of a session (Session Storage), or beyond (Local Storage). With this technology, web applications can store a much greater amount of data compared to HTTP cookies. The storage capacity provided by web storage varies from 5MB to 25MB, depending on the browser.

The specification also includes a client-side JavaScript API, known as the Web storage API, which is needed in order to manage the data store and its content. An innovative feature of this technology is that a web application can use this API to retrieve locally stored information even when the browser is offline.

Web storage is a technology that is completely based on client-side scripting. Unlike HTTP cookies, data cannot be sent via HTTP headers. The web storage API is the only way in which a web application can access the data in web storage.

Similarly to HTTP cookies, the security model of web storage is also based the same-origin policy. This means that each origin has a unique storage object assigned to it. For this reason, the specification does not recommended using this technology on websites that are using a shared host name. Moreover, it recommends treating persistently stored data as potentially sensitive, as it could contain information such as email addresses or calendar appointments. It also recommends using Web Storage on website that are served over

HTTPS, in order to avoid information leakage or spoofing, in case, for example, of DNS spoofing attacks.

As in the case of HTTP cookies, a third-party tracking agent could use Web Storage to profile users across multiple sessions (WHATWG, 2017). The specification recommends browser vendors to treat web storage content in the same manner as they treat HTTP cookies. In particular, vendors are encouraged to organise the user interfaces for clearing data in a way that allows users to clear all different types of persistent data simultaneously.

While Web Storage is a much lesser known technology than HTTP cookies, its usage is not exempt from regulations around personal user data.

2.3. WEB SQL DATABASE

Web SQL database (Hickson, 2009) is a specification that allows web applications to store large amounts of data in the browser, using client-side transactional databases that can be queried using SQL. The specification is based on SQLite, a relational database management system developed by D. Richard Hipp. The peculiarity of SQLite is in the fact that it is an embedded database system, meaning that it is not built using the usual client-server pattern, but it is instead part of the client application (Owens 2006).

Since the beginning of 2010 a few browser vendors started implementing experimental versions of the Web SQL database API (Chromium Blog, 2010). However, this was not a complete novelty for some of them. Web SQL Database stores data in a very similar way to Google Gears, and both technologies are based on SQLite. Other browser vendors, instead, decided to avoid Web SQL database completely. Mozilla, in fact, refused to implement this technology in Firefox, explaining the reasoning behind this choice in a blog post by Ranganathan (2010).

The main concerns raised by Ranganathan lied in the de facto adherence of Web SQL Database with SQLite. He argued that it was not appropriate to derive a speciation from an already shipped technology. For example, since SQLite did not implement a clearly defined version on SQL, drafting a specification around it would mean having to refer directly to the SQLite manual. Ranganathan was also concerned that web developers and browser vendors would end up being bounded to a third-party technology that could evolve independently from the HTML standards. He advocated for the adoption of a “web native” JavaScript API that could act as an abstraction between the web application and the underlying technology used by the browser to store data.

In November 2010, the W3C followed suit and announced the decision to abandon the Web SQL Database draft, citing that that “all interested implementers have used the same SQL backend (SQLite)”, and lamenting the lack of multiple independent implementations (Hickson, 2010). Web SQL Database was deprecated in favour of Indexed Database API.

Despite the deprecation by the W3C, three major browser vendors (Chrome, Safari and Opera) have continued supporting Web SQL Database and have not yet announced any plan of discontinuing it.

2.4. INDEXED DATABASE API

The first draft of this specification was initially published under the name of WebSimpleDB API (Mehta, 2009) and it was renamed to Indexed Database API the following year (Mehta 2010). It defines a JavaScript-based interface for an embedded transactional database system. Similarly to Web Storage and Web SQL database, IndexedDB allows storing structured information in the browser and the API provided is the only interface a web application needs to access and manipulate data. The main difference in comparison to Web Storage is in the scale and structure of the data that can be stored. In fact, Web Storage provides a basic key-value store that can be useful when dealing with simple datasets. On the other hand, Indexed Database API allows storing larger amounts of structured data and it provides advanced features such as in-order retrieval of keys and storage of duplicate values for a key.

In the context of Indexed Database API, a database is made of one or more object stores, which are the elements responsible for holding the data. Databases are identified by the origin they are associated with, by a name and by a version number. An object store is the primary storage mechanism for storing data in a database. It comprises of a list of records and each record consists of a key and a value. Object stores are identified by a name, which is unique in the context of the database. An index allows looking up records in an object store using properties of the values. When a new database is created, it is empty, meaning that it does not contain any object store. A database version change is required in order to change the set of object stores. In order to efficiently retrieve records stored in an indexed database, each record is organized according to its key. Operations such writing and reading data are performed using requests, each of them need to placed against a transaction. Transactions are responsible for interacting with the data in a database and execute requests (Alabbas and Bell, 2017).

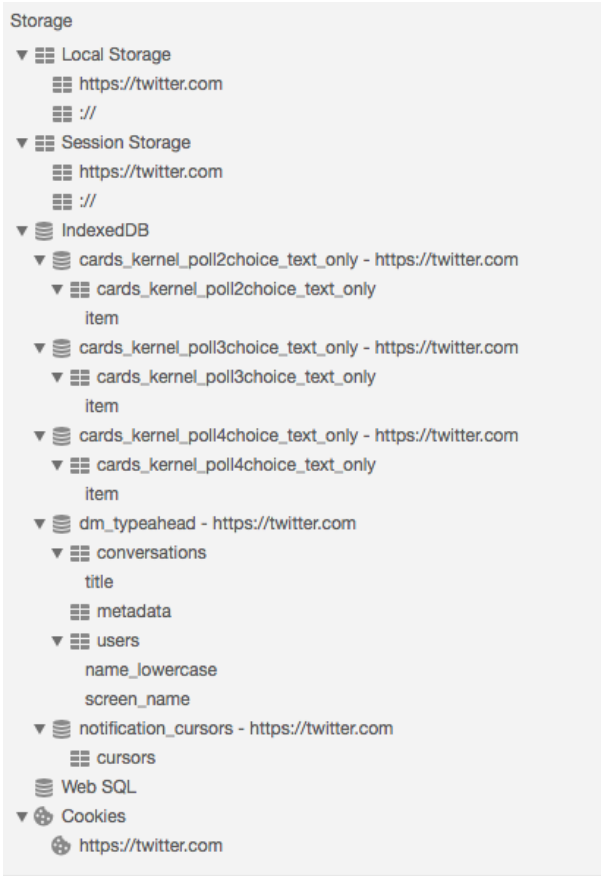

Figure 1 – Representation of client-side stored data provided by the console of Chrome.

Figure 1 shows the content of IndexedDB data, alongside content saved by other client-side storage mechanisms, as shown from the console of Chrome. It can be noted that IndexedDB can store data in a much more structured manner compared to HTTP cookies or Web Storage. In fact, the screenshot shows that several databases are associated to the same origin. Each database has one or more object stores and the content of each can be sorted through one or multiple keys

Unlike WebSQL Database, IndexedDB is an object-oriented database. The interface for adding and retrieving data does not use SQL queries but keys and indexes instead. Nonetheless, one of the principles that were considered while designing the IndexedDB API, was to allow it to be easily wrapped by JavaScript libraries built on top of it (Ranganathan and Wilsher 2010). This means that the primitives of IndexedDB could be used to create a CouchDB-style API or even an SQL-based API.

The security recommendations for the usage of Indexed Database API are not different to those for Web Storage. The security model of IndexedDB still gravitates around the principles of the same-origin policy. This means that browsers assign a set of databases to each requesting origin, based on a combination of the hostname, the port number and the protocol used by the web application. A web application is allowed to access locally stored data as long as the origin of the request matches the origin of the local database. Unlike HTTP cookies, a maximum storage duration does not have to be specified.

The screenshot displayed in Figure 2 shows how unencrypted IndexedDB content can be easily accessed using a browser's default debugging tool. This example is taken from a major news publisher and it is particularly interesting as the content displayed on the console is supposed to be hidden behind a pay wall. While the usual user interface (on the left-hand side) successfully blocks non-paying users from reading the article (labelled as 'premium'), its content can be read via the console on the right-hand side.

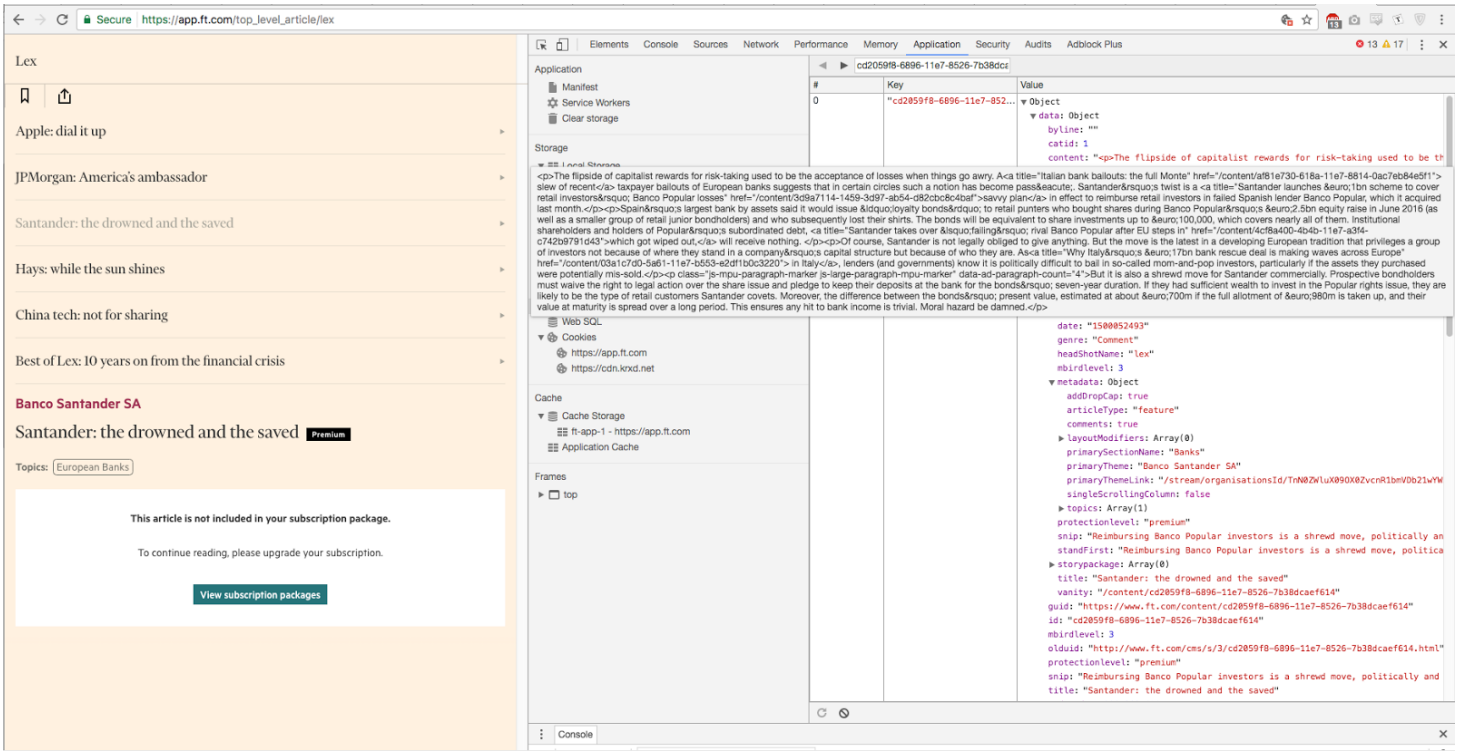

Figure 2 – Example of restricted data that can be accessed via the application console - Retrieved in July 2017.

3. LITERATURE REVIEW

Security concerns around in-browser data storage systems have been raised since the early days of HTML5. This chapter summarises a selection of papers that focused on Web Storage, Web SQL Database and Indexed Database API from a cybersecurity prospective. The reader will note that some of the issues highlighted in these works pre-exist these three technologies and have, in fact, been around since the introduction of HTTP cookies. However, it is clear that the enhanced capacities of these new client-side storage systems have broadened the scope of the security risks associated to web applications dealing with user data.

There seem to be three main themes around which the literature has focused its attention so far. The first focuses on the integrity and the confidentiality of user data stored in browser. The works relate to questions such as "how can an attacker get hold of personal information" or "can users and developers trust this technology, especially in the context of the same origin policy?" The second theme is around client-side storage used as a vector to track users across different websites and sessions. Compared to the first theme, the difference is that the focus is not on preventing an attacker from stealing sensitive information, but rather on monitoring how third-party trackers operate in the wild. The third theme is intimately related to the second, and focuses on preventing tracking agents from identifying users and collecting personal information against their will.

The following paragraphs will discuss a selection of papers that touch upon those three themes, putting particular attention to those studies that cover Web Storage, Web SQL Database and Indexed Database API. Some essential works that relate to adjacent technologies such as HTTP cookies will also be covered.

3.1. CONFIDENTIALITY AND INTEGRITY OF LOCALLY-STORED CONTENT

Hanna et al. (2010) performed a real-world study of client-side storage mechanisms. In their work, they analysed several web applications that make use of primitives for client-side storage and they highlighted the risks of persistent client-site cross-site scripting attacks. They discovered that the data saved in client-side storage systems is frequently used by the web application without being sanitised. The authors list several examples of this practice occurring in major websites. The lack of sanitisation opens the gates to the possibility of a new type of attack, which is persistent in its nature. Indeed, an attacker could manage to insert a malicious payload into a client-side storage system through well-known attack channels, such as a transient cross-site vulnerability or by modifying the packets destined to the victim over the network. When the compromised local data is used by the web application in any code evaluation construct, the payload has all the potentials of a malicious script hosted in the local database. The script would be able to remain in the local database until the data store is cleared, an event that in most cases happens very rarely. In this scenario, an attacker would only need to access the storage system once and be able to compromise a client application over a prolonged period of time. Furthermore, the authors also pointed out that the server is likely to remain unaware that the attack ever occurred. The types of client-side storage primitives that Hanna et al. (2010) considered are localStorage (part of Web Storage), Web SQL Database and Google Gear (both now deprecated). In order to mitigate the risks of a stored XSS attack, they suggested that bowsers should "automatically remove any potentially executable script constructs inside database values before returning them".

Similarly**, Tump (2011)** provides an example of how a persistent client-side XSS attack can be performed using Web SQL. Taking advantage of poor input validation an attacker could insert a malicious script into a client-side database, and get it to execute every time the data is retrieved. In the same work, the author also discusses other common risks related to

client-side data storage. While, according to Tump (2011), the risk of client site SQL injection seems to be negligible, there are some other security issues that might arise. The author mentions the risk of client-side data corruption and data leakage, which can occur when an attacker gets hold of the user's file system. The author also highlights the risks associated in relying on the same origin policy. First, browsers often fail to properly enforce the same origin policy. Second, sometimes web application developers fail to specify the origin correctly, for example ignoring the possibility that the same policy can apply to third level domains they do not control. Third, a DNS poisoning attack allows an attacker to bypass the same origin policy and access all the personal information stored in the browser for a given origin.

Shah (2012) listed WebSQL in the to 10 security threats introduced by HTML5. The author reiterates the argument that if a web application is vulnerable to cross-site scripting attacks, an attacker could retrieve information from the in-browser database and transfer it across domains. The paper also demonstrates that this type of exploit can be achieved even without any prior information about the database schema.

Lekies and Johns (2012) discuss some security issues related to the usage of Web Storage as a method for caching fragments of mark-up or client-side scripts. They describe three possible attack scenarios that could facilitate the insertion of a malicious payload in Web Storage: cross-site scripting, man-in-the-middle attack and the usage of shared browsers. The authors also perform an analysis on the usage of Web Storage in the wild, by crawling the front pages of the Alexa top 500,000 websites and analysing them with the Web testing framework HTMLUnit. They found that about 4% of the sites analysed make use of Web Storage. Among them, the authors discovered that Web Storage is used in a potentially unsecure way in at least 5% of the cases. Finally, the authors propose a mitigation mechanism that relies on verifying the integrity of the data stored in Web Storage, before adding its content to the DOM. This approach is then evaluated in the context of the three attack scenarios mentioned above.

Preuveneersa et al. (2013) discuss the feasibility of HTML5 web applications in the context of e-health. Whilst the potential benefit of persistent storage techniques such as Web SQL Database and Web Storage is acknowledged, one of the main concerns raised is the lack of data encryption.

Jemel and Serhrouchni (2014) propose a system to encrypt content stored using localStorage in which each user is assigned to a specific storage space. In their model, users are prompted to specify an identifier and a password in order to access the data stores. Those credentials are then used to encrypt personal data. The advantage of this approach is that it could allow sharing information that relate to the same user across multiple machines, in a format that is encrypted on the client.

Englehardt et al. (2015) study the possibility of passive eavesdropping related to the usage of third-party tracking cookies. They show that an attacker could observe the value of HTTP cookies in transit and use them to reconstruct a significant amount of the user's browsing history. This is possible because most web pages contain a variety of third-party cookies, which allow identifying a given user regardless of their IP address. In particular, the work shows that an adversary could reconstruct between 62% and 73% of a user's browsing history. The authors suggest that this practice is likely to be used as tool of mass surveillance by state-level intelligence agencies such as the National Security Agency (NSA) and the Government Communication Headquarters (GCHQ).

Kimak (2016) argues that Indexed Database API, in its current state, is "unavoidably insecure" due to its design. Similarly to other studies, the main concern raised is that data is stored in unencrypted format and therefore vulnerable to attacks such as cross-site scripting. The author also proved that personal data could be retrieved using forensic tools, even after a

user requests its deletion via the browser's user interface. Despite the security concerns, the author acknowledges the advantages of IndexedDB, especially in terms of performance. The work proposes a security model that relies on the usage of encryption and multi-factor authentication. The suggestion is to always encrypt data in the database with a key that is stored on a remote server. If the client wants to access the local data, it would have to make a server-side request to decrypt it. Before releasing a session key identifier, the server will have to authenticate the user via multi-factor authentication. This approach can provide a greater level of data protection, however, one of its limitations is that it requires an Internet connection in order to decrypt the data, thus losing off-line support, which is one of the key features of IndexedDB.

Sendiang et al. (2016) documented a case study of a PHP-based web application showcasing few server-side techniques to reduce the risk on SQL injections. The web application makes use of Indexed Database API to store sanitised user information before sending it to the server. While they seem to suggest that the usage of IndexedDB is mostly advantageous from a performance point of view, it is also worth noting that their work can be seen as an example of how a client side database might act as a buffer between the user and the server. The presence of IndexedDB between the mark-up and the server can, in fact, help enforcing input validation and data sanitation, protecting the server from interacting with malformed data.

On a similar note, an earlier paper, written at the time when the specifications were still being drafted, seem to provide a much more positive outlook on the usage of client-side storage. **Hsu and Chen (2009)** suggest that, despite the well-know concerns, storing information in the browser not only provides a performance advantage, but also allows users to have greater control over their data. Equally, since data are not stored in a centralised data centre, the scope for data breaches can be narrower, compared to a traditional web application. Moreover, the authors note that, since technologies such as Web Storage (localStorage) do not require data to be sent via HTTP headers, the possibility for a packet sniffing attack is reduced, compared to the case of HTTP cookies.

More recently, **Tsalis et al.** (2017) conducted a study that evaluates the protection offered by private browsing mode. The authors analyzed four major desktop browsers and uncovered instances of privacy violations that contradict the browsers' own documentation. In particular, they discovered that all of the browsers analyzed leave traces of user activity in the file system, even after a private browsing session is concluded. This means that a malicious party that takes control of a user's machine could retrieve sensitive information related to previous private browsing sessions. The situation is aggravated by the fact that this type of information is stored in the same folders used for other type of browsing data storage and caching. This makes the process of retrieving data related to private sessions a trivial task, which could be performed even by a malicious party who is oblivious of the potentials of digital forensic tools. Moreover, the authors point out that the documentation provided by browser vendors suggests that all data will be discarded at end of the private browsing session. This could generate in the user a false sense of privacy protection. The paper also proposes and evaluates a mitigation measure that entails storing information in a virtual file system, based on volatile memory rather that on a hard drive. The authors demonstrate that off-the-shelf software can be used to create a virtual file system, which can guarantee data deletion at the end of the browsing session. Moreover, they show that it can be combined with file-shredding software to allow a more granular selection of the content to discard.

3.2. CLIENT-SIDE STORAGE SYSTEMS AS TRACKING VECTORS

Krishnamurthy and Wills (2009) conducted a study on the diffusion of private user information performed by third-party trackers that use a combination of HTTP cookies and other elements of the DOM. The authors looked for the presence of requests to third-party domains on a selection of 1200 popular websites and collected statistical data over a period of four years. One key aspect of the methodology used is that, in order to identify third-party requests, the authors verify the domain name and compare the authoritative DNS servers used by both the third-party resource and the page that hosts it. This approach allows to correctly classify cases in which code belonging to the same tracking provider is served over different endpoints using various DNS CNAME hosts or via a content delivery network. The results showed that the collection of user data increased over time, in particular the latest period analysed (September 2008) showed a penetration of 70%. This trend was also noticed in 'fiduciary sites', in which the user is expected to provide confidential information such as medical or financial details. Moreover, they noted that 52% of the websites considered contained code from at least two third-party tracking entities. In contrast, it was noticed that data collection was performed by an ever-decreasing number of actors. In fact, during the period considered, several company acquisitions occurred in the tracking industry, allowing a small number of companies to significantly expand their reach. This allowed them to easily track the user's activities across the majority of the popular websites. This work is also significant because it highlights that, in the wild, the boundaries between first-party and third-party tracking code are often not clear. This is due to the emergence of tracking techniques that entail a combination of first-party cookies and third-party JavaScript libraries. The authors suggest that this approach seems to be primarily used by sites involved in analytics, meaning that those trackers would not necessarily create cross-site browsing profiles. However, they do not exclude that trackers involved in behavioural tracking might also resort in using a similar approach.

Soltani et al. (2009) conducted a study on the usage of Flash Local Shared Object as a tracking vector, which is often referred to as 'Flash cookies'. They analysed the top 100 domains ranked by QuantCast. On 31 of them, they found at least a case of data overlap between HTTP cookies and Flash cookies, meaning that a same value was appearing on the data stored in both systems. Moreover, they witnessed several occurrences of what they defined as "cookie respawning", in which the value of a deleted HTTP cookie is restored in the background, taken from a Flash cookie that keeps a back-up copy of it. On a follow-up study, **Ayenson et al. (2011)** observed the emerging usage of Web Storage (localStorage) as a tracking vector. While they did not find that this storage system was directly employed as part of respawning mechanisms, they noticed several cases of matching values among HTTP cookies and Web Storage data, which they named 'HTML5 cookies'.

"Evercookie" is a technique presented by **Kamkar (2010)** that significantly increases the resilience of tracking HTTP cookies. The author describes evercookies as "cookies that won't go away". The mechanism consists of a client-side API that replicates the HTTP cookie data across several types of client-side storage systems. The documentation lists more than a dozen of them, including Flash Local Shared Object (Flash cookies), Web Storage (mentioned as HTML5 Session Storage, HTML5 Local Storage and HTML5 Global Storage), Web SQL Database and Indexed Database API. This multi-tier approach allows personal information to remain in the browser even after a user manually removes HTTP cookies. In fact, once the evercookie API detects that a HTTP cookie has been deleted, it can recreate it in the background, using the information stored in any of the other vectors available. As long as the information is maintained in at least one of the many storage systems used, the evercookie API is able to restore it in all the others. The result is a zombie cookie that is extremely hard to remove even for advanced users. Moreover, the Evercookie API can also synchronise tracking data across different browsers, through plug-ins such as Flash (via Local Shared Object), Silverlight (via Isolated Storage) or Java (via JNLP PersistenceService or the CVE-2013-0422 exploit documented in Oracle, 2013).

Roesner el al. (2012) presented an in-depth investigation of web tracking performed by third-party actors. The work analysed a corpus of around 1000 websites, spanning from very popular to lesser-used websites, and found the presence of over 500 unique trackers. The authors propose a classification of trackers that goes beyond the usual notion of first-party and third-party trackers. Instead, they propose a classification system based on the tracking behaviour that is observable from the client (Table 1). This system also challenges the significance of classifying cookies as either third-party or first-party. In fact, all cookies are first-party in the context of their own origins and often users visit those origins as 'first-party clients', such as in the case of social networks. For this reason, the authors suggest the usage of the terms like "tracker-owned" cookies and "site-owned" cookies.

Table 1 - Classification of trackers proposed by Roesner et al. (2012)

Class	Description	Type of cookies	Notes
Third-Party Analytics	Provides an analytics library to individual websites	Site-owned	Site-owned cookie value is leaked to tracker's domain for collection
Third-Party Advertising	User tracking for the purpose of targeted advertising	Tracker-owned	User never visits tracker's domain directly
Third-Party Advertising with Popups	Similar to third-party tracking, but using a popup window as a mechanism to circumvent third-party cookie blocking	Tracker-owned	User is forced to visit the tracker's site by a popup window
Third-Party Advertising Networks	Tracking code inserted indirectly by another cooperating tracker	Tracker-owned	Relies of information passed by the coopering tracker
Third-Party Social Widgets	Tracking code set by social media sites users generally have an account with	Tracker-owned	User voluntarily creates an account with the site

The analysis also documents the occurrence of "cookie leaks", in which the contents of a cookie associated to a given origin are passed as parameters in a request to another origin, with the purpose of circumventing the browser's same-origin policy. Furthermore, the work also quantifies the usage of alternatives to HTTP cookies. The authors found 'remarkably little use' of Web Storage (localStorage). In fact, out of the 524 trackers identified, they found that this storage mechanism is used in only 8 cases. Moreover, in only 5 of them it was found to contain unique identifies. In all of those 5 cases, the user identifiers were a copy of the values found on HTTP cookies. These cases are instances of cookie respawning. Flash LSOs were used by 35 trackers, but only on 9 cases duplicate content to HTTP cookies was identified. Other aspects of this work will also be discussed on paragraph 3.3 of this dissertation.

Acar et al. (2014) performed a large-scale analysis of a selection of advanced persistent tracking mechanisms. They reported the usage of Indexed Database API as a storage mechanism of tracking data, albeit in a small number of cases (20 out of the 100 000 analysed - 0.02%). Indeed, they found one instance in which IndexedDB data matched the content of a Flash Local Stored Object. The authors claimed that this was the first time a research paper documented evidence of the usage of IndexedDB as an evercookie vector.

The specifications of the three storage systems considered in this work all recommend browser vendors to represent the different storage systems in a way in which users can

associate them to HTTP cookies, particularly in terms of data deletion. However, studies such as **West and Pulimood (2012)** reported that on some browser the operation of removing local databases is not straightforward.

Some works in the literature discuss the practice of cookie matching (or cookie syncing), a technique that is used in real-time advertising bidding, which allows trackers to associate different tracking profiles that relate to the same user. **Olejnik et al. (2013)** quantifies both the frequency and the breadth of data leakage related to cookie matching. They analysed a sample of 100 user profiles and found that 91 of them were subject to cookie matching, showing instances of trackers leaking 27% of a users' browsing history. They also prove that users with a longer tracking history are more valuable to advertisers than newcomers. Moreover, they show that the market value of parts of a users' browsing history can be as low as a fraction of a US dollar cent.

Englehardt (2014) also discusses cookie-syncing, warning that it can allow sharing of personal data between different tracking servers, a practice of which the users have very little visibility. Cookie syncing can also further enhance the impact of cookie respawning. In fact, while most major trackers do not use mechanisms such as the aforementioned evercookie, they might share user information with trackers that do use techniques of cookie resurrection. The author warns that the practice of cookie syncing, combined with other sophisticated mechanisms used by trackers, makes it almost impossible for a user to start from a completely new browsing profile.

Bujlow et al. (2015) conducted a comprehensive literature review of the tracking mechanisms developed in previous years. The three systems analysed in this thesis are mentioned in their work as possible tracking vectors, under the category of storage-based tracking systems. Amongst their findings, the authors noticed that Safari seems to carry over user data from a normal-mode user identity in a private browsing mode, which is an undesirable feature, as it does not protect the user from being tracked.

Derksen et al. (2016) discuss the usage of Web Storage (localStorage) and Indexed Database API in the context of tracking. The authors analyzed the behavior of twenty popular tracking services on a selection of about a thousand websites. They found that localStorage was used by 15% of the trackers analyzed. Moreover, none of the website analyzed showed the usage of Indexed Database API as a tracking vector. They nonetheless did not exclude that IndexedDB could be used in circumstances different from those considered in their work. One of the security concerns they raised, is that expiry dates are not supported by either technology, which could lead to private information being stored for longer time compared to HTTP cookies. The authors also study the implementation of data deletion and find that the browsers analyzed allow the deletion of Web Storage (localStorage) and IndexedDB data via the same user action needed to remove cookies. Similarly, **Bujlow et al. (2015),** seem to imply that the content of data stored using these techniques is automatically emptied at the time when the cookies are cleared. This dissertation, however, will show that, in a different set of major browser, data deletion requires an extra user step in order to include HTML5-related client-side storage techniques (see Chapter 5).

Wu et al. (2017) presented an evaluation of private browsing mode in desktop and mobile browsers in the context of fingerprinting. The work highlights many inconsistencies in the way private mode is implemented across different browsers and also between the desktop and mobile versions of the same browser. One of the findings of their study is that all the browsers considered, delete all client-side stored data created in private mode after the user closes the private session. **Derksen et al. (2016)** also presented a similar finding in their work. In contrast, Chapter 5 of this dissertation will show that this is not always the case on all major browser. It will provide a few examples of browsers where data gathered during a private session is not deleted after the user terminates the private session.

Gonzalez et al. (2017) performed a large-scale study on the usage, content and format of HTTP cookies in the wild. Their work analyses a large dataset of network data that comprises of 5.6 billion HTTP requests. First, they performed an analysis on the reach of cookies, by measuring the number of different referrers that originate a HTTP requests to the same cookie-setting endpoint. They found that, while the vast majority of cookies relate to a unique referrer domain, there is a long tail of cookies whose originating requests come from a significantly high number of different domains. This means that the breadth of certain major trackers is quite vast. In the dataset considered, the authors noticed the presence 212 cookies that could reach more than 10000 domains. The authors also analyzed the usage of cookie names in the wild. They found instances of websites that use cookies whose names include a unique identifier of the user. The study also considered the content and format of cookies used in the wild. It showed that some cookies tend to grow in length as the user visits more sites. This is because there are instances in which a single cookie can be used to concatenate different type of information, often using complex schemas that deviate from the basic key-value pair structure (name=value). In a few extreme cases, different data was combined in the same cookie value using JSON format. The authors also found that in one third of the cases analyzed, the value of cookies contained a datestamp mixed with other information. This means that those cookies could contain unique identifiers, despite appearing as different at a first sight. The work presented by Gonzalez et al. is significant because it challenges some of the assumptions taken in previous works, not only on the current usage of cookies *per se* but also with regards to the methods used to identify tracking cookies. Moreover, the study found instances of cookies values containing personal identifiable information such as users' IP and email address, which, of course, represent a serious breach of privacy.

3.3. PREVENTIVE MEASURES AGAINST USER TRACKING

The client-side storage mechanisms mentioned above can be all used to identify and track users across different browsing sessions and webpages. In 2007, several public interest groups wrote a letter to the United States Federal Trade Commission expressing their desire to create a '**Do Not Track list**' for online advertisers (Schwartz et al., 2007). The name was chosen after the 'Do Not Call Registry, database maintained by the United States federal government, listing the telephone numbers of individuals who have requested that telemarketers not contact them (Federal Trade Commision, 2017). The original proposal would have entailed that ad provides submit their information to the FTC, which would compile a list of the domain names used by trackers. Browser vendors could then subscribe to this list, and give their users to effectively block many forms of tracking.

Two researches, Soghoian and Stamm, implemented a similar feature, albeit following a different approach, in which the client communicates the user's preference with regards to tracking via an HTTP header **(Soghoian, 2011)**. According to this approach, tracking agents are expected to refrain from identifying users and perform their usual activities according to the preference expressed by the DNT header. This proposal was extremely impactful and most major browser implemented the 'Do Not Track' header by the following year. Moreover, in 2015, the W3C started the work of formalising this feature into a web standard called Tracking Preference Expression (DNT) **(Fielding and Singer, 2017)**.

Mylonas et al. (2013) analysed the security controls of several mobile and desktop browsers. The work highlights that desktop browsers generally provide better protection, as the controls available for desktop users are greater that those available on mobile browsers. The analysis found that users of the mobile browsers considered do not have the option to opt-out of third-party cookies. The authors also pointed out that support for the 'Do Not Track' header is unavailable on most of the mobile browsers analysed and found that in some cases the user interface that deals with security features can be confusing. Moreover, the authors found a number of security issues on two major mobile browsers.

Mayer (2011) studied a series of technologies developed to protect users from third-party trackers. The author found that community-maintained blocklists are the most effective way to prevent undesired user tracking. Those lists mainly consist of URL blacklists and they are generally used in conjunction with browser extensions, such as AdBlock Plus (eyeo GmbH, 2017). The author also claims that tracking is often inextricably tangled with third-party advertising, therefore often blocking trackers also entails blocking code that provides advertisements.

Similarly**, Kontaxis and Chew (2015)** describe a tracking protection feature of Mozilla Firefox, whose approach is based on the same mechanism used by ad-blocking browser extensions such as AdBlock Plus. It analyses all outgoing HTTP requests and matches them against a blacklist, which is based on a curated list of tracking origins. Requests to known tracking domains are blocked and an icon appearing next to the URL bar notifies the user. They evaluated this approach on a dataset consisting on 200 popular news sites. They found that this feature blocks at least one unsafe element on 99% of sites tested. An increase in page performance was also noticed, thanks to a 44% median reduction in page load time and a 39% median reduction in data usage. Furthermore, in a second experiment conducted using data sent by Firefox Nightly users, the authors found that 14.1% of page loads contain at least one tracking element.

Blacklists are also a powerful tool to protect users from phishing and malware. **Virvilis et al. (2015)** provide a comparison of the protection measures against rogue sites available on desktop and mobile browsers. The work commences by acknowledging that mobile browsers often offer a lower level of protection compared to their desktop-based counterparts and in some cases they offer no protection at all. The authors propose a browser-agnostic mitigation approach that overcomes the technical limitations related to each specific browser and the user's natural inertia in downloading browser extensions or dealing with often confusing security settings. This novel strategy, named 'Secure Proxy', consists of a HTTP forward proxy that operates at network level to filter content before it is transmitted to the user's device. The filtering mechanism is delegated to a third-party service that assesses the reliability of the content providers, based on the aggregation of multiple blacklists and AntiVirus engines.

Building from the previous work, **Nisioti et al. (2017)** revisits the anti-phishing mechanisms available for users of mobile browsers of three popular operating systems. The study reveals that the protection against rogue websites provided by pre-installed web browsers is still very poor and in some cases null. The situation seems a bit better in the case of third-party browsers, such as Firefox or Chrome on Android. The authors, however, note that installing third-party browsers require an active action from the user. Moreover, in the iOS ecosystem, neither the default browser nor third-party browsers offer protection against phishing attacks. In this context, the authors discuss an extension of 'Secure Proxy' and propose TRAWL (TRAnsparent Web protection for alL). Similarly to 'Secure Proxy', TRAWL is implemented outside the users' device in order to avoid resource consumption and to be usable across different platforms and browsers without requiring any custom installation. The tool provides DNS and URL filtering based on a collection of curated blacklists, but it does not delegate the filtering mechanism to a third-party service. In fact, in this approach the filtering logic is handled locally, which prevents the leaking of user information and also overcomes the limitation of usage rate limits.

The work by **Roesner el al. (2012),** (mentioned in paragraph 3.2) also reviews the defence strategies against user tracking available to browser users. First, they point out that blocking third-party cookies is not an effective method because some browsers only block the operation of setting a third-party cookie, but not access to its content. This would not prevent a tracker from reading the value of a cookie containing user-identifying information, after this value was set on a previous visit to social media sites or by advertising popups. Second, the

authors notice that the 'Do Not Track' header does not seem to have any visible effect in preventing tracking. Indeed, it is a policy that relies on the goodwill of the tracker. Moreover, it appears that many of the parties involved with user tracking argue that their behaviour should not be considered tracking as it is defined by the DNT specification, and consequentially refuse to implement it. The authors also mention the limitations of private browsing mode, noting that it is primarily designed to protect users from attackers with physical access to the machine and not necessarily from remote user tracking. As a method of protecting users' privacy, the authors propose ShareMeNot, a browser extension that limits third-party tracking code that belongs to social media sites, while making sure that actual functionality visible to the user remains unaffected. In practice, the extension allows tracking requests to be sent only when the user clicks on an embedded social media button (such as Facebook's "Like"). The solution proposed by the authors has been subsequently incorporated into another privacy tool named "Privacy Badger".

Privacy Badger is a browser extension that follows an alternative approach to curated blacklists. It uses algorithmic methods to decide which resource is tracking the user and verifies whether scripts that belong to a given domain collect unique identifiers even after sending a "Do Not Track" message. If so, it automatically disallows content from that third-party tracker (Privacy Badger, 2017).

The practice blocking of user tracking is often strictly related to the blocking of adverting content. **Nithyanand et al. (2016)** performed a study on the counter-blocking mechanisms implemented by publishers to prevent the loss of income associated to the usage of ad-blocking techniques. The authors analysed 5000 popular websites and found that anti-ad-blocking mechanisms were used in 6.7% of the cases. The anti-ad-blocking techniques that can be used generally entail adding bait advertising elements to the DOM and periodically verifying their presence or content. If ad-blocking is detected the hosting website would return a message to the users asking them to disable any ad-blocking tool and, in some cases, prevent the users from further navigating to the site.

4. USAGE OF CLIENT-SIDE STORAGE IN THE WILD

This section discusses the methodology for an investigation on the usage of a selection of client-side storage techniques in the wild. The investigation has two main objectives. The first objective is to quantify the frequency of the usage of these techniques on a large-scale sample of the World Wide Web. This is useful since they all are relatively new techniques and the necessity to quantify their usage in the wild has been mentioned on other works in the literature (Kimak, 2016). Moreover, it will provide a context for the second objective, which is to quantify the pervasiveness of these techniques in the context of third-party tracking code.

4.1. METHODOLOGY

The analysis is performed by inspecting the content of a large-scale dataset containing snapshots of client-side scripts used by websites, representing a significant portion of the World Wide Web. In this task, the content of those scripts is parsed, with the purpose of finding instances of certain JavaScript constructs that suggest the usage of a certain primitive. Figure 3 shows a high-level diagram of the tools and resources that were used, and the following sections will provide an in-depth explanation of each one of its parts.

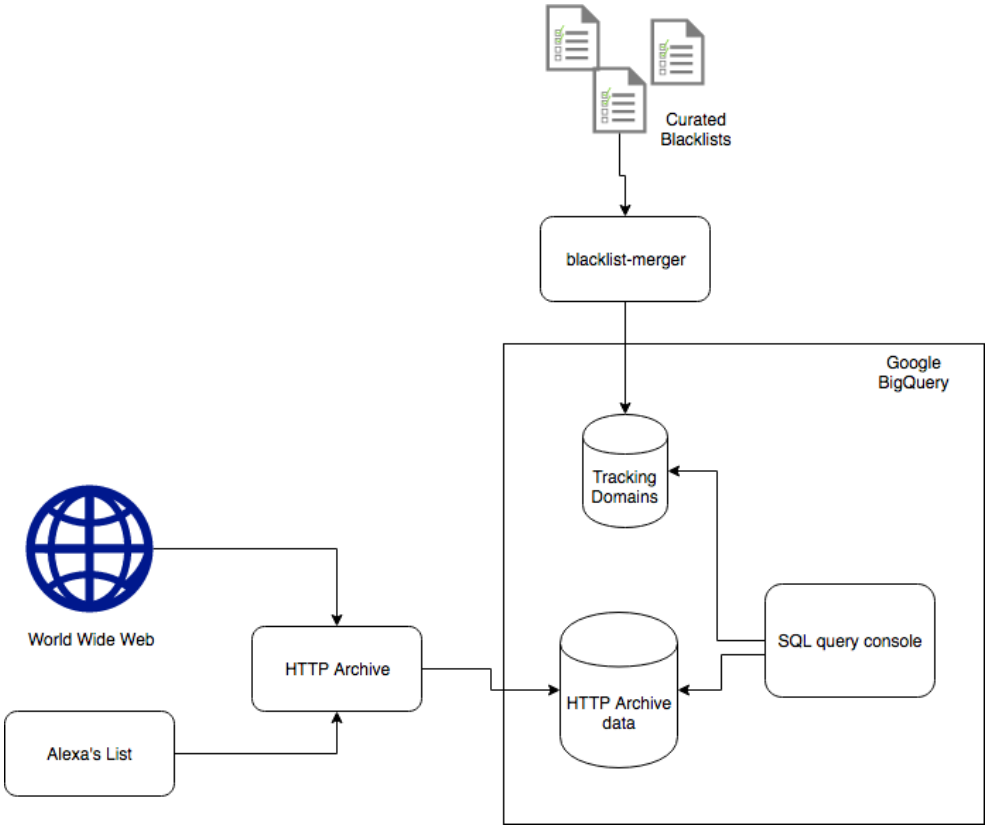

Figure 3 - Architecture diagram of the tools and systems involved

4.1.1. Tools and resources used

There are a few open-source tools that can be used in order to perform an extensive audit, even with a relatively limited amount of resources. **HTTP Archive** is a project created by Souders (2010). Every fortnight, it crawls a list of webpages and archives information such as the payload content and the logging of the interaction between the browser and the crawling

client. The list of the webpages crawled by HTTP Archive is based on the Alexa Top 1,000,000 Sites (Alexa Internet, 2017). HTTP Archive also captures the body of the responses for each subresource used by the website scanned. The scans are performed with two Chrome user agents: one for desktop and one Android (emulated).

The datasets generated by HTTP Archive can be freely downloaded and they can also be used in conjunction with big data tools such as **Google BigQuery** (Grigorik, 2013). The latter is a web service that allows the analysis of large datasets, using SQL. This suits the needs of this project, since the size of the datasets generated by HTTP Archive can be of up to several hundreds of gigabytes.

4.1.2. Determining the matching rules

As previously mentioned, the core idea of this investigation is to quantify the presence of certain constructs in the dataset provided by HTTP Archive. In order to do so, for each one of the primitives considered, a matching rule was been defined by listing some of the constructs required to perform basic operations, such as creating a data store, reading and writing data.

The **Web Storage** API provides two storage mechanisms, one for handling data within a current session (sessionStorage) and another one that lasts beyond the current session (localStorage). In this work, only the constructs used by localStorage were considered, as sessionStorage would not be relevant due to its transient storage mechanism. Table 1 shows the constructs needed in order to read or write data using localStorage.

Table 2 - Constructs used by Web Storage (localStorage)

Web Storage (localStorage) constructs	
localStorage	Property of the 'window' object that needs to be used to access the Storage assigned to each origin
setItem	Method of the Web Storage API that adds a new item to the storage
getItem	Method of the Web Storage API that retrieves item to the storage

The same process was followed for the **Indexed Database API**. The constructs mentioned in Table 2 are part of the steps necessary to create a local database containing an object store and to access the store to either read or write data.

Table 2 - Constructs used by Indexed Database API

IndexedDB API constructs	
indexedDB	Attribute of the 'window' object that provides applications a mechanism for accessing IndexedDB (of type 'IDBFactory')
transaction	Method of the IndexedDB API needed to access the object store
objectStore	Method of the IndexedDB API that returns an object store in the scope of the transaction

Similarly, table 3 shows the constructs necessary to read and write data using the now deprecated **Web SQL Database** API.

Table 3 - Constructs used by Web SQL Database (deprecated)

Web SQL Database (deprecated) constructs	
openDatabase	Method that opens a Web SQL database, or creates a new one if none is found
transaction	Method of Web SQL API needed to access the database
executeSql	Method of the Web SQL API that defines the SQL command to perform in a given transaction

The constructs identified above were used to define the matching rules used to query the dataset. This approach is based on the assumption that if a webpage contains a certain combination of those constructs in any of its subresources, it is likely to use the APIs considered to store data locally. For example, a JavaScript snippet that saves data using the Indexed Database API needs to contain all three constructs mentioned in table 1 ('indexedDB', 'transaction', 'createObjectStrore'). Similarly, writing or reading data with Web Storage (localStorage) requires the 'localStorage' construct and either a 'setItem' or 'getItem' construct. Table 4 shows the matching rules defined for each primitive.

Table 4 - Matching rules used for each of the primitives analysed

Primitive	Matching rule
Indexed Database API	"indexedDB" AND "transaction" AND "objectStore"
Web Storage (localStorage)	"localStorage" AND ("setItem" OR "getItem")
Web SQL database (deprecated)	"openDatabase" AND "transaction" AND "executeSql"

4.1.3. Querying the dataset

The rules defined in the section above can be used to create a series of SQL queries, which can be run against the HTTP Archive dataset using Google BigQuery. The code snippet in example 1 is a simple query that filters all subresources of the dataset according to the matching rule defined for the Web Storage API (localStorage). The queries used in this work follow the Standard SQL Syntax adopted by Google BigQuery (Google Cloud Platform, 2017), which is compliant with the SQL:2011 standard (International Organization for Standardization, 2011).

```
SELECT
      *
FROM
      `httparchive.har.2017_09_15_chrome_requests_bodies`
WHERE
```

```
REGEXP_CONTAINS (body, r'localStorage')
AND (REGEXP_CONTAINS(body, r'setItem')
    OR REGEXP_CONTAINS(body, r'getItem'))
```

Example 1 – Simple query that returns subresources containing Web Storage (localStorage) constructs

4.1.4. Trackers list

This paragraph discusses the creation of a database of tracking hostnames, based on open-source blacklists of trackers. The purpose of the database is to help identifying which subresources in the HTTP Archive dataset are involved with user tracking. There are several curated blacklists of trackers available on the web, the ones considered in this work are: Disconnect (2017), No Track (Quidsup, 2017) and Easy List (2017).

Creating a database of tracking hostnames is necessary because the content of blacklists cannot be used in its original format for the purpose of this work. This is due to the fact that tracking blacklists are part of third-party software, such as ad-blocking browser extensions. They often contain constructs that belong to the browser extension itself, like comments or delimiters. Moreover, different blacklists use different schemas. For example, the dataset used by Disconnect is in JSON format (Example 2), whereas East List makes use of multiple text files combined to form a lengthy regular expression (Example 3). Therefore, to be usable in the context of this work, data needs to be sanitised and converted to a consistent format.

```
"ADP Dealer Services": {
      "http://www.adpdealerservices.com/": [
          "admission.net",
          "adpdealerservices.com",
          "cobalt.com"
      ]
}
```

Example 2 – Extract from the tracking blacklist used by Disconnect Me

```
||audiencerate.com^$third-party
||auto-ping.com^$third-party
||autoaffiliatenetwork.com^$third-party
```

Example 3 – Extract form the tracking blacklist used by Easy List

Consistency of the data format is not the only issue. Most blacklists often contain whitelists too, which are exceptions to the blocking rules that are needed in order to workaround specific use cases[1]. To avoid the occurrence of false positive, it is necessary

[1] For example, certain AdBlock rules are too strict for some websites and, if respected, would result in loss of core functionalities.

make sure the content of whitelists is not included in the database. This can be done by cherry picking only certain sections of a blacklist list from its open-source repository (Table 4).

Table 5 - Blacklists considered and relative repositories

Black list	List repository
Disconnect Tracking Protection	https://raw.githubusercontent.com/disconnectme/disconnect-tracking-protection/master/services.json
Easy Privacy	https://raw.githubusercontent.com/easylist/easylist/master/easyprivacy/easyprivacy_trackingservers.txt https://raw.githubusercontent.com/easylist/easylist/master/easyprivacy/easyprivacy_trackingservers_international.txt
No Track	https://raw.githubusercontent.com/quidsup/notrack/master/trackers.txt

Since tracking lists are updated on a regular basis, it is important to make sure that the trackers database contains data that is relevant to point in time in which the HTTP Archive scan considered was generated. To facilitate the process of creating and updating the trackers database, an automated tool, "Blacklist-merger", was developed as part of this work. Blacklist-merger extracts the domain names from the lists mentioned in Table 4 and performs a process of data sanitisation, which involves removing comments or any construct other than the tracking hostname. Once the data is sanitised, it is collected in an array, where duplicates are removed. Finally, the array is used to create a comma-separated values file (Figure 4) that can be uploaded to Google BigQuery and matched against the HTTP Archive table.

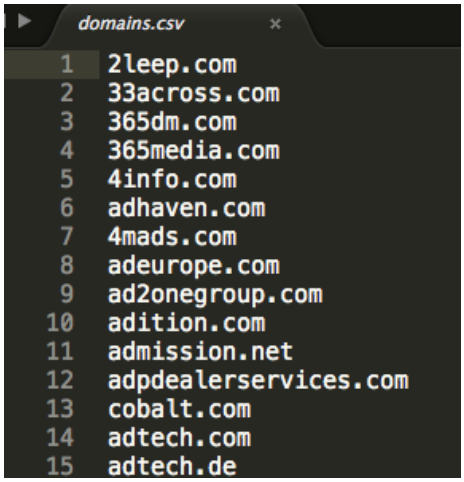

Figure 4 - Example of CSV file generated by 'blacklist-merger'

4.1.5. Workflow

The final step is to combine all the elements mentioned in the previous sections, and query the dataset using the Google BigQuery interface. Appendix 8.1.1 shows the query created for this work, which returns a table of statistical data for the HTTP Archive dataset. Moreover, a filter that limits the results to the Alexa top 10000 websites was created, with the purpose of verifying whether the output of the query would differ if applied to the a smaller selection of most popular websites.

4.1.6. Limitations

The approach taken in this research has a few limitations, which deserve to be mentioned. Some of them are related to the dataset chosen, others to the nature of the methodology itself. First, the scanning engine used by HTTP Achieve truncates payloads that are greater than 2 MBs. This means that if the constructs defined in the matching rules happen to be in the part of the payload that HTTP Archive could not capture, they will not be found by the query. Payloads greater than 2MBs are, however, a rare occurrence, as highlighted by the percentage of truncated subresources, in the results section.

A further limitation related to the usage of HTTP Archive is that it can only provide snapshots of front pages of openly available websites. The scanning engine does not perform operations such as user log in or following links on a menu. Considering that primitives such as the Indexed Database API are designed to support advanced web applications, it is reasonable to assume that there are cases of websites in which those storage techniques are used only once the user is logged in. However, this is an accepted limitation, especially considering that in order to quantify the usage of client-side storage techniques in the context of user tracking, it is far more important to focus on the large-scale adoption of the technologies in question rather than specific use cases.

In addition, HTTP Archive does not contain snapshots from each one of the Alexa Top one million sites. The set of websites scanned is loosely based on the Alexa list, but any private individual could send a request to HTTP Archive to add or remove sites to the dataset. For this reason, the analysis was performed on both the whole dataset and on a selection of it that contains only the Alexa top 10000 sites. The actual number of website included in each scan is specified in the results section.

Lastly, this type of analysis verifies the presence of certain given constructs in client-side scripts, but it cannot verify the actual usage of the primitives. For example, a website could include a JavaScript library that relies on Web Storage, but never execute its code in the browser. Moreover, some websites include third-party libraries that perform a set of basic operations using a given primitive with the sole purpose of assessing browser capabilities. This practice is known as ‘feature detection’ and one of the most well-known libraries used for this purpose is Modernizr (Ateş et al., 2017).

4.2.1. Usage of the primitives considered

This section presents the results returned by running a set of queries similar to the one shown in Appendix 8.1.1, against the dataset provided by HTTP Archive for the 1st of December 2017. The HTTP Archive dataset provides data for about half a million websites and it contains the payload of more than 16 million subresources, which include HTML documents, style sheets and JavaScript files (Table 6). The percentage of truncated subresources can be considered as a margin of error, as it represents the subresources on which the matching rules are not applicable.

Table 6 - Characteristics of the dataset used

Number of websites in the dataset	431851
Total number of subresources in the dataset	16167545
Truncated subresources (%)	0.03

Table 7 shows the usage of the primitives considered, on the whole dataset. An interesting result is that more than two thirds of the websites analysed contain Web Storage related constructs. Another result worth noticing is that the constructs analysed are very often found on third party subresources.

Table 7 - Results for the whole dataset

	Websites with construct in subresource (%)	Websites with construct in 3rd party subresource (%)	Subresources containing matching construct (%)
Web Storage (localStorage)	70.30	63.91	6.66
Indexed Database API	5.28	4.91	0.16
Web SQL Database	1.51	1.23	0.04

The same set of queries can be used, with some small variations (Appendix 8.1.2), to filter the scan to the Alexa top 10000 sites. This reduces the scope of the analysis to around 8,500 websites and approximately 400,000 subresources (Table 8).

Table 8 – Subset of the dataset including only the Alexa top 10k websites

Number of Websites in the dataset	8518
Total number of subresources in the dataset	419009
Truncated subresources (%)	0.05

Table 9 shows the results for the Alexa's 10000 sites. It is interesting to notice that, after applying the filter, the values for the usage of the Indexed Database API are almost double compared to the whole dataset.

Table 9 – Results for the Alexa top 10K

	Websites with construct in subresource (%)	Websites with construct in 3rd party subresource (%)	Subresources containing matching construct (%)
Web Storage (localStorage)	82.05	75.64	8.03
Indexed Database API	10.24	8.96	0.26
Web SQL Database	2.56	1.96	0.06

4.2.2. Usage of the primitives as a tracking vector

This section highlights the usage of the same three client-side storage techniques in the context of user tracking. As it can be seen in Table 10, there is a high percentage of websites containing at least one tracking subresource were constructs that belong to Web Storage (localStorage) can be found. The figures are much smaller for Indexed Database API and considerably smaller for Web SQL Database.

Table 10 – Websites and tracking subresources

Websites with at least one tracking subresource using the primitive considered (%)	Full dataset	Top 10k
Web Storage (localStorage)	55.39	65.20
Indexed Database API	2.24	4.51
Web SQL Database	0.84	0.83

Table 11 highlights the usage of the client-side storage techniques in the context for tracking from a different angle. It shows the percentage of subresources that belong to tracking domain amongst all the subresources containing the constructs for the primitives considered. In other words, this table answers the question: "how frequently are those storage techniques used as tracking vectors?" In all cases, the figures seem to be surprisingly high, starting from around 40% for Indexed Database API to almost 70% for Web Storage (localStorage). This is a significant finding because it shows that user tracking is a major use case for those primitives. Equally, it is significant to find that this is also the case for a deprecated standard such as Web SQL Database.

Table 11 - Tracking subresources and primitives

Subresources using the primitive considered that are flagged as 'tracker' (%)	Full dataset	Top 10k
Web Storage (localStorage)	69.80	64.18

Indexed Database API	43.68	45.13
Web SQL Database	53.50	31.95

5. USER CONTROL OVER LOCALLY STORED DATA

5.1. METHODOLOGY

This section will try to answer the following questions: What type of control do users have over data that is stored in the browser using mechanisms such as Indexed Database API? Are web browsers respecting the recommendations and do they make it easy for users to delete the content of client-side storage?

This section also presents Storage Watcher, a lightweight web application created for this dissertation project. The application can be used to verify how web browsers implement Web Storage (localStorage), Indexed Database API and Web SQL Database, particularly focusing on how effectively a user can clear locally stored data. The application uses the primitives mentioned above to create a data store and populates it with some dummy data.

5.1.1. Architecture

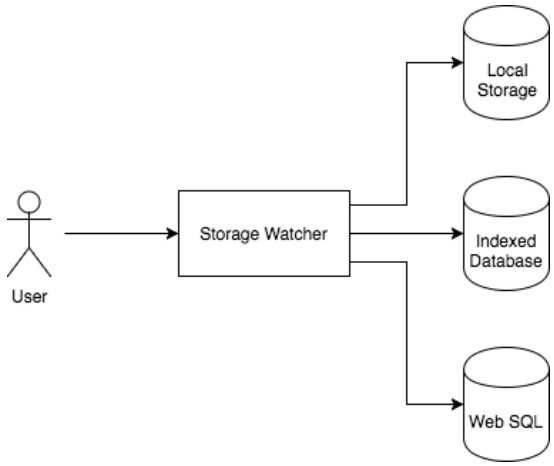

Figure 5 – High-level architecture of Storage Watcher

The architecture of the application is presented in Figure 5. Storage Watcher is designed to perform two tests: one to verify the level of API support of a given browser, the other to verify the effectiveness of data deletion.

5.1.2. User Interface

Storage Watcher contains a JavaScript library, written as part of this work, which creates three client-side data stores, using the three primitives mentioned above. The library creates the data stores and defines their schema. The user interface allows populating the data stores with some dummy data and listing the content of the data stores. Table 11 lists the read and write operations that can be performed using the user interface.

Table 12 - Operations performed by Storage Watcher

Operations	Notes
List all entries	Database/storage read. On the app's default state all storage systems should contain no entries
Add one entry	Database/storage write

Using Storage Watcher, it should be possible to notice that the content of the database remains consistent across different browsing sessions. The application does not need to be online in order to operate.

5.1.3. Browser support test

Most browsers broadly support the three client-side storage primitives considered in this work. There are, however, a few exceptions. For example, Firefox disables the Indexed Database API when a user is browsing in private browsing mode. The desired outcome of this task is to provide a comprehensive overview of how different browsers support the primitives considered. Storage Watcher can be used to perform a manual test that verifies whether each of the API considered is enabled. The test consists of a sequence of simple write and read operations and the steps necessary to perform it are described in Table 12. Moreover, on page load, Storage Watcher performs a check to verify if the properties of the primitives considered are present in the Document Object Model. If they are not, an alert will be displayed to the user. In that case performing the write/read test manually will not be necessary; as Storage Watcher would have already detected that the API is not supported.

Table 13 - Browser support test

Steps	Expected behaviour
1. Click on 'Add entry'	A pop-up alert should inform the user of the success of the write operation.
2. Click on 'List all entries'	A series of pop-up alert should list all the entries stored.

5.1.4. Effectiveness of data removal

As mentioned in the previous chapters of this report, the specifications recommend browser vendors to treat the data removal of various client-side persistent data features in the same way as HTTP cookies. This means that browsers are expected to make it easy for users, or at least possible, to remove all locally stored user data. Storage Watcher can be used to perform a test that verifies whether and to which extent different browsers respect this requirement. Table 13 shows the steps required to perform this test.

Table 14 – Steps required to perform the data removal test

Steps	Expected behaviour and notes
1. Click on 'Add entry'	A pop-up alert should inform the user of the success of the write operation.
2. Close the current tab	This is needed to ensure the current session is terminated.
3. Remove personal data	This should be done using the user interface of the browser (e.g. clicking 'clear browsing data' in Google Chrome). Note that this step is not needed if the test is performed in private browsing mode.
4. Click on 'List all entries'	No pop-up alerts should be expected.

After deleting all personal data, a returning user should expect the local content related to Storage Watcher to be in its default state and to contain no entries. When clicking on 'List all entries', if any entry can still be found, it means that the local content has not been reset as part of the data removal process.

5.2. RESULTS

This paragraph discusses the results of tests conducted using Storage Watcher. The tests were performed in November 2017, on a broad selection of browsers, including Firefox, Chrome, Safari, Opera, Edge/Internet Explorer on different platforms. Table 15 and Table 16 show the full results for each combination of browser, operating system and device considered.

Table 15 – Results for desktop browsers

				API support			Data deletion		
Platform	**OS**	**Browser**	**Mode**	**localStorage**	**IndexedDB**	**Web SQL**	**localStorage**	**IndexedDB**	**Web SQL**
Desktop	Mac OS 10.12.5	Firefox 57.0 (quantum)	default	supported	supported	not supported	data deleted	***data deleted only if 'Offline website data' is expicitly selected by the user***	N/A
			private	supported	***disabled***	not supported	data deleted	N/A	N/A
		Firefox 56.0	default	supported	supported	not supported	data deleted	***data deleted only if 'Offline website data' is expicitly selected by the user***	N/A
			private	supported	***disabled***	not supported	data deleted	N/A	N/A
	Mac OS 10.10.5	Chrome 62	guest	supported	supported	supported	***data deleted only after quitting chrome***	***data deleted only after quitting chrome***	***data deleted only after quitting chrome***
			default	supported	supported	supported	data deleted	data deleted	data deleted
			incognito	supported	supported	supported	data deleted	data deleted	data deleted
		Opera 49	default	supported	supported	supported	data deleted	data deleted	data deleted
			private	supported	supported	supported	data deleted	data deleted	data deleted
		Safari 10.1.1	default	supported	supported	supported	data deleted	data deleted	data deleted
			private	***disabled***	supported	***disabled***	N/A	data deleted	N/A
	Windows 10	Edge 40	default	supported	supported	not supported	data deleted	data deleted	N/A
			inPrivate	supported	***disabled***	not supported	data deleted	N/A	N/A
		Chrome 62	default	supported	supported	supported	data deleted	data deleted	data deleted
			guest	supported	supported	supported	***data deleted only after quitting chrome***	***data deleted only after quitting chrome***	***data deleted only after quitting chrome***
			incognito	supported	supported	supported	data deleted	data deleted	data deleted
		Firefox 56	default	supported	supported	not supported	data deleted	***data deleted only if 'Offline website data' is expicitly selected by the user***	N/A
			private	supported	***disabled***	not supported	data deleted	N/A	N/A
		Opera 49	default	supported	supported	supported	data deleted	data deleted	data deleted
			private	supported	supported	supported	data deleted	data deleted	data deleted
	Windows XP	Internet Explorer 11	default	supported	supported	not supported	data deleted	data deleted	N/A
		Chrome 62	default	supported	supported	supported	data deleted	data deleted	data deleted
			incognito	supported	supported	supported	data deleted	data deleted	data deleted
		Firefox 47	default	supported	supported	not supported	***data deleted only if 'Offline website data' is expicitly selected by the user***	***data deleted only if 'Offline website data' is expicitly selected by the user***	N/A
		Firefox 56	default	supported	supported	not supported	data deleted	***data deleted only if 'Offline website data' is expicitly selected by the user***	N/A
		Firefox 57	default	supported	supported	not supported	data deleted	***data deleted only if 'Offline website data' is expicitly selected by the user***	N/A
			private	supported	***disabled***	not supported	data deleted	N/A	N/A

Table 16 – Results for mobile browsers

Platform	OS	Browser	Mode	API support			Data deletion		
				localStorage	IndexedDB	Web SQL	localStorage	IndexedDB	Web SQL
Mobile	iOS 10.2.1	Safari	default	supported	supported	supported	data deleted	***data perists after clearing local data***	data deleted
				disabled	supported	***disabled***	N/A	data deleted	N/A
		Chrome 62.0	default	supported	supported	supported	data deleted	***data perists after clearing local data***	data deleted
			incognito	***disabled***	supported	***disabled***	N/A	data deleted	N/A
		Firefox 10.2	default	supported	supported	supported	data deleted	data deleted	data deleted
			private	***disabled***	supported	***disabled***	N/A	data deleted	N/A
		Opera 16	default	supported	supported	supported	data deleted	data deleted	data deleted
			Mini	not supported	not supported	not supported	N/A	N/A	N/A
	iOS 11.1.2	Safari	default	supported	supported	supported	data deleted	data deleted	data deleted
			private	***disabled***	***disabled***	***disabled***	N/A	N/A	N/A
		Firefox 10.3	default	supported	supported	supported	data deleted	data deleted	data deleted
			private	***disabled***	***disabled***	***disabled***	N/A	N/A	N/A
		Chrome 62.0	default	supported	supported	supported	data deleted	data deleted	data deleted
			private	***disabled***	***disabled***	***disabled***	N/A	N/A	N/A
		Opera 16	default	supported	supported	supported	data deleted	data deleted	data deleted
			private	supported	supported	supported	***data perists after closing private session***	data deleted	data deleted
			Mini	not supported	not supported	not supported	N/A	N/A	N/A
	Android 6.0	Firefox 57	default	supported	supported	not supported	data deleted	***data perists after clearing local data***	N/A
			private	supported	not supported	not supported	data deleted	N/A	N/A
		Firefox Focus 2.4	default	supported	supported	not supported	data deleted	data deleted	N/A
		Chrome 62.0	default	supported	supported	supported	data deleted	data deleted	data deleted
			incongnito	supported	supported	supported	data deleted	data deleted	data deleted
		Opera 43.0	default	supported	supported	supported	data deleted	data deleted	data deleted
			private	supported	supported	supported	data deleted	***data perists after closing private session***	***data perists after closing private session***
		Opera Mini 31.0	default	not supported	not supported	not supported	N/A	N/A	N/A
		Microsoft Edge Preview 1.0.0	default	supported	supported	supported	data deleted	data deleted	data deleted
			inPrivate	supported	supported	supported	data deleted	data deleted	data deleted
		MiuiBrowser 9.1.3	default	supported	supported	not supported	***data perists after clearing local data***	***data perists after clearing local data***	N/A
			incognito	supported	***carries over values from non incognito version***	not supported	***data perists after closing private session***	***data perists after closing private session***	N/A
		Edge 1.0	default	supported	supported	supported	data deleted	data deleted	data deleted
			inPrivate	supported	supported	supported	data deleted	data deleted	data deleted
	Windows Phone 8.10 by HTC	Internet Explorer	default	supported	supported	not supported	data deleted	***needs extra step: "advanced settings" > "manage storage"***	N/A
	Android 7.0	Firefox 57	default	supported	supported	not supported	data deleted	***data perists after clearing local data***	N/A
		Opera 43.0	private	supported	supported	supported	data deleted	***data perists after closing private session***	***data perists after closing private session***

5.2.1. API support

There are a few peculiarities in the way browsers support the client-side storage APIs considered that deserve to be highlighted. Firefox and Edge, for example, disable the Indexed Database API when used in private browsing mode. In both cases, other storage techniques remain available. In contrast, certain versions of iOS WebKit-based browsers (Safari, Chrome and Firefox for iOS) and Firefox for Android, seem to do the exact opposite, as they disable the Web Storage and Web SQL Database APIs when in private mode, but not the Indexed Database API.

If the reasoning behind disabling certain client-storage APIs is to prevent user tracking, it is important to remember that advanced tracking mechanisms, such as the ones mentioned in the literature review, employ multi-tier approaches based on a combination of various storage vectors. Therefore, blocking certain APIs whilst allowing the usage of others might not

produce the desired level of privacy. It is, however, worth mentioning that more recent versions of iOS-WebKit-based browsers have introduced a more consistent approach on which all the three APIs are disabled on private browsing mode.

Lastly, when running the test on MiuiBrowser 9.1.3, it was noticed that the browser carries over the values of Indexed Database API content created while using the application on standard browsing mode. Taken in the context of a private browsing session preceded by a regular usage of the browser in its standard mode, this browser behaviour could allow a third party tracker to resume and recreate tracking values set while the user was browsing on previous non-private sessions and identify them even if they are browsing in private mode.

5.2.2. Effectiveness of data removal

The right-hand side of Tables 14 and 15 show that the process of removing private data from a browser does not always delete information stored in all of the three client-side storage techniques considered in this work. In particular, certain versions of iOS-WebKit-based browsers (Safari and Chrome for iOS) and some Android browsers (Firefox for Android and MiuiBrowser) retain Indexed Database API content even after a user requests data deletion. In all the cases considered, the user interface not only does not make clear that Indexed Database API content will persist but also give the impression that all 'offline web site data' will be deleted (Figure 6). Furthermore, in MiuiBrowser 9.1.3, Web Storage (localStorage) content is also maintained, after requesting the deletion of private data.

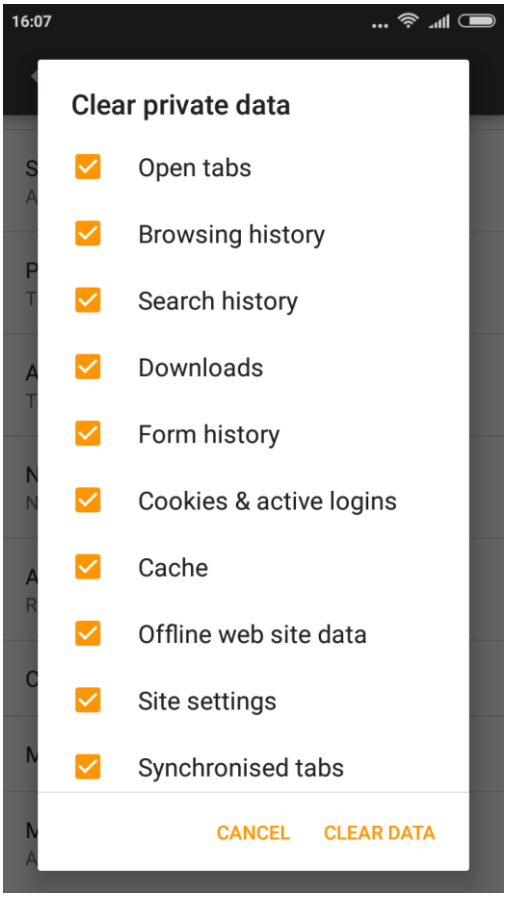

Figure 6 - Firefox 57 on Android 6.0. The user interface suggest that offline data will be removed.

At least in the case of iOS browsers, this issue seems to be resolved in the latest version of the software considered in this work. However, this behaviour can still be seen on other recent browsers (Firefox 57 on Android 7.0).

It is also worth pointing out that some browsers require the user to perform an extra action in order to include Indexed Database API content to the process of clearing private data. As a matter of fact, on all the desktop versions of Firefox considered, whilst the user interface allows deleting data stored via Indexed Database API using the same panel used to remove HTTP cookies, this option is disabled by default. This means that a users would have to expand the 'details' dropdown and manually add 'offline website data' if they wish to remove Indexed Database API content at same time of HTTP cookies. On an earlier version of Firefox analysed (Firefox 47 on Windows XP), this was also the case for Web Storage (localStorage). This default setting could be misleading for an inexperienced user and give a sense of anonymity that cannot be guaranteed, especially considering that, as mentioned before, the Indexed Database API could be used as a backdoor to reinstate content of HTTP cookies.

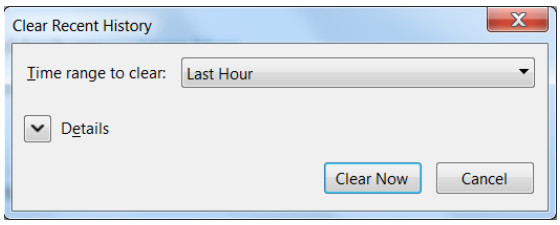

Figure 7 – Panel used to remove personal information, including HTTP cookies from Firefox. The option to remove content stored using Indexed Database API is disabled by default.

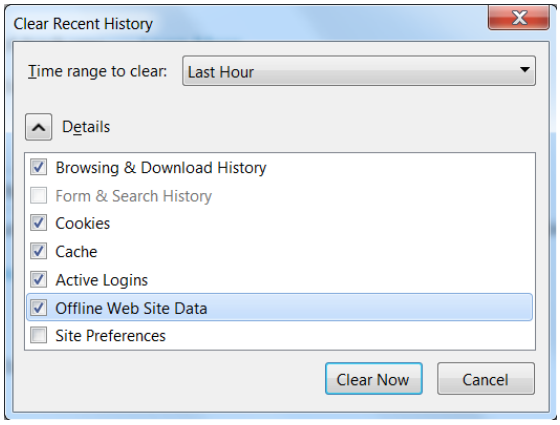

Figure 8 - In order to remove Indexed Database API content users need to manually add it from the options list.

Similarly, Internet Explorer Windows Phone 8.10 by HTC requires a separate action to remove Indexed Database API content. In this case, the user needs to navigate to a different menu item called "advanced settings" and, from there, choose the option "manage storage".

5.2.3. Persistence of data across private browsing sessions

Running this experiment has also highlighted that some browsers do not fully isolate client-side stored data when used in private mode. According to the results of this test, Opera 43 on Android persists data stored using Indexed Database API and Web SQL Database across different private browsing sessions. In contrast, the same behaviour was noticed in Opera for iOS, but in the case of Web Storage (localStorage). Similarly, it was noticed that MiuiBrowser 9.1.3 persists data stored on both Web Storage (localStorage) and Indexed Database API across different private browsing session.

One final note is about Google Chrome guest mode: running this test has shown that content stored in each of the three APIs considered is persisted across different windows opened in guest mode. This means that a user would need to quit Chrome completely in order

to discard locally-stored data accumulated in a guest browsing session. This behaviour might be misleading for certain users who might assume that simply closing the browsing window but not the application, as in the case of a private browsing window, might be enough to remove locally-stored private data.

6. CONCLUSIONS & FUTURE WORK

This dissertation analysed three client-side storage techniques that have been part of the ecosystem of front-end development for almost a decade. In the first chapters of this report, it was shown, through various citations, that the three primitives considered were initially designed to enrich the user experience of everyday websites and make them behave like native applications without the need of third-party plugins. However, this work has shown that user tracking seems now to be a major use case for these technologies. This section recapitulates the key findings of the work performed for this dissertation and presents a few ideas for further work.

6.1. CLIENT-SIDE STORAGE AS A TRACKING VECTOR

Section 4 of this work presented a large-scale analysis on the usage of client-side storage techniques as tracking vectors in the wild. To the best of the writer's knowledge, this is the first comprehensive study that focuses on the adoption of Web Storage, Indexed Database API and WebSQL API, in the context of user tracking. It could be argued that analysing the real-world usage of these three primitives, regardless of their usage as tracking vectors, is already a significant contribution. Indeed, a previous work lamented the lack of usage statistics for browser-based storage systems (Kimak, 2017). The only study found on this subject is a much earlier work (Lekies and Johns, 2012), which estimated that the adoption of Web Storage (localStorage) is around 5% of the dataset considered. The work proposed in this dissertation has provided an up-to-date audit and it showed that the percentage of sites congaing constructs that relate to Web Storage (localStorage) is considerably higher than the previous study have found (70.30% and 82.05%, respectively for the whole dataset and the top 10k sites). The work also showed that the adoption of the three storage systems considered is inhomogeneous. In fact, compared to Web Storage (localStorage), the percentages of usage for Indexed Database API are much lower (5.28% and 10.24%) and closer to its deprecated predecessor Web SQL Database (1.51% and 2.56%).

With regards to the occurrence of the three storage systems in tracking scripts, this work showed that a significant number of websites contain at least one tracking subresource containing code construct that belong to Web Storage (localStorage): 55.39% for the whole dataset and 65.20% for the top 10k sites. Figures are much lower for both Indexed Database API (2.24% and 4.51%) and Web SQL Database (0.84% and 0.83%). More importantly, this work has shown that tracking scripts seem to currently be the major employers of the three storage primitives considered. Indeed, in all three cases, the subresources that contain the construct analysed are often identified as trackers. In the case of Web Storage (localStorage) the figures are 69.80% for the whole dataset and 64.18% for the top 10k sites, 43.68% and 45.13% for Indexed Database API and 53.50% and 31.95% for Web SQL Database.

Other works, presented in Chapter 3, have analysed the usage of Web Storage (localStorage) and Indexed Database API by trackers. To briefly recapitulate, Roesner el al. (2012) found that 8 of the 524 trackers analysed make use of Web Storage (localStorage). Acar et al. (2014) showed the usage of Indexed Database API as tracking vector, albeit in a small fraction of the cases considered (0.02%). Derksen et al. (2016) found that Web Storage (localStorage) is used by 15% of the trackers considered, but found no evidence of the usage of IndexedDB in the dataset considered.

However, both the methodology and the findings of this study are different. First, this dissertation performs a large-scale analysis covering almost half a million websites. Second, it analyses the content of code snippets rather than performing an actual page visit. Third, it

relies on curated blacklists to identify trackers, as opposed to behavioural analysis. Moreover, the prospective of the results provided in this work is novel. Instead of measuring the pervasiveness of browser-based storage by comparing its occurrence to adjacent technologies such as HTTP cookies, this work quantifies the use case of tracking against the total usage of the primitives considered.

Furthermore, to the best of the writer's knowledge, no other work quantified the usage of Web SQL Database by trackers. This is presumably due to the fact that the technology itself is deprecated. Despite this, this work has shown that constructs belonging to this primitive can still be found in the wild, albeit at slightly lower rates than its successor, IndexedDB.

As mentioned in paragraph 4.1.6, one of the limitations of the methodology used is that it detects the presence of certain code constructs, but it cannot verify their actual usage. This is a limitation of static code analysis of JavaScript and it is due to its dynamic nature. As future work, it could be worth developing a methodology to further validate the findings of this work. It will be, however, necessary to consider the complexities highlighted in Gonzalez et al. (2017), in particular, around the way unique identifies can be bundled in a unique value with other data.

6.2. IMPORTANCE OF CONSITENT DATA REMOVAL

Chapter 5 presented a study that assessed how browsers support the removal of data stored using the three storage systems considered in this work. This was done using Storage Watcher, a web application that was specifically built for this dissertation. In this analysis, a broad range of browsers and devices were considered and it was proved that not all browser allow to easily clear data from client-side storage systems. Moreover, this work showed cases of data leakage between standard and private browsing mode. These findings are significant because they suggest that users of the affected browsers are more exposed than others to the risk of undesired tracking. Indeed, to avoid the 'resuscitation' of tracking data, browsers need to allow users to remove all data at the same time. At the current status this is not easily achievable, even on some major browsers.

Other works, discussed in Chapter 3, have touched on the topic of data deletion. As mentioned, Derksen et al. (2016) also analysed data deletion of Web Storage (localStorage) and IndexedDB content. The scope of Derksen et al. (2016) is, however, limited to four desktop devices. In contrast, a significant finding of this thesis is that data deletion is not fully supported on some mobile browsers.

More recently, Wu et al. (2017) performed an analysis of data persistence in private browsing mode and did not find any instance of either Web Storage or IndexedDB data preserved across different private browsing sessions. This thesis has, however, considered a larger range of devices and has shown that instances of data leakage outside private browsing mode can still be found on some major browsers.

In terms of future work, it would be worth extending the Storage Watcher application in order to analyze the case of client-side storage associated to third-party origins. In particular, it would be interesting to verify if and how browsers allow users to prevent data being stored by third-party domains. Moreover, it could be worth verifying if there is any difference in the way data deletion is implemented in the case of first-party and third-party origins.

7. REFERENCES

Alabbas, A., Bell J., 2017, Indexed Database API 2.0, W3C Proposed Recommendation, 16 November 2017, https://www.w3.org/TR/IndexedDB-2/

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

Adobe Systems, 2012, *What are local shared objects?* in Security and privacy, http://web.archive.org/web/20121230094342/http://www.adobe.com/security/flashplayer/articles/lso/

Alexa Internet, Inc., 2017, *Alexa Top 1,000,000 Sites*, [online database] http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

Anthes, G., 2012. HTML5 leads a web revolution. Communications of the ACM, 55(7), pp.16-17.

Ateş, F., 2017, *What is Modernizr?*, https://modernizr.com/docs/#what-is-modernizr

Ayenson, M.D., Wambach, D.J., Soltani, A., Good, N. and Hoofnagle, C.J., 2011. Flash cookies and privacy II: Now with HTML5 and ETag respawning. https://www.truststc.org/education/reu/11/Posters/AyensonMWambachDpaper.pdf

Barth, A., 2011a, RFC 6265. HTTP State Management Mechanism. IETF, 2011. https://tools.ietf.org/html/rfc6265

Barth, A., 2011b. The web origin concept, 2011. IETF RFC6454. https://tools.ietf.org/html/rfc6454

Bujlow, T., Carela-Español, V., Solé-Pareta, J. Barlet-Ros, P., 2015. *Web tracking: Mechanisms, implications, and defences*. arXiv preprint arXiv:1507.07872. Vancouver https://arxiv.org/pdf/1507.07872.pdf

Can I Use, 2017. IndexedDB API support comparison. Available at *http://caniuse.com/#search=indexeddb*

Chromium Blog, 2010, More resources for developers, https://blog.chromium.org/2010/01/more-resources-for-developers.html

Constantin, A., 2016, *Cookie limits - the server side story* in Alin Constantin's Blog. http://alinconstantin.blogspot.hk/2016/11/cookie-limits-server-side-story.html

Cox, P., 2011, *Top 10 Reasons to Use HTML5 Right Now*, in Codrops https://tympanus.net/codrops/2011/11/24/top-10-reasons-to-use-html5-right-now/Google Code, 2011 http://gearsblog.blogspot.hk/

Derksen, I., Poll, I.E., van den Broek, F., 2016. *HTML5 Tracking Techniques in Practice*. http://www.cs.ru.nl/bachelorscripties/2016/Ivar_Derksen___4375408___HTML5_Tracking_Techniques_in_Practice.pdf

Disconnect, 2017, *Who we are* , https://disconnect.me/about

Easy List, 2017, *About*, https://easylist.to/pages/about.html

Englehardt, S., 2014. The hidden perils of cookie syncing. Freedom to Tinker. https://freedom-to-tinker.com/2014/08/07/the-hidden-perils-of-cookie-syncing/

Englehardt, S., Reisman, D., Eubank, C., Zimmerman, P., Mayer, J., Narayanan, A. and Felten, E.W., 2015, May. *Cookies that give you away: The surveillance implications of web tracking*. In Proceedings of the 24th International Conference on World Wide Web (pp. 289-299). International World Wide Web Conferences Steering Committee.

Etzioni, A., 1999. *Privacy isn't dead yet*. The New York Times, http://www.nytimes.com/1999/04/06/opinion/privacy-isn-t-dead-yet.html.

eyeo GmbH, 2017, *Getting started with Adblock Plus*, https://adblockplus.org/getting_started#general

Federal Trade Commission, 2017. National Do Not Call Registry. URL: https://www. donotcall. gov

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T., 1999. *Hypertext transfer protocol--HTTP/1.1* (No. RFC 2616).

Hanna, S., Shin, R., Akhawe, D., Boehm, A., Saxena, P. and Song, D., 2010. *The emperor's new APIs: On the (in) secure usage of new client-side primitives*. In Proceedings of the Web (Vol. 2). http://www.comp.nus.edu.sg/~prateeks/papers/w2sp10-primitives.pdf

Hickson, I., 2010. Web sql database. *W3C, Editor's Draft*.

Hickson, I., 2011. Web sql database. *W3C Working Group Note 18 November 2011* https://www.w3.org/TR/2010/NOTE-webdatabase-20101118/

Hickson, I., Hyatt, D., 2008. HTML 5, A vocabulary and associated APIs for HTML and XHTML W3C, Working Draft 22 January 2008, https://www.w3.org/TR/2008/WD-html5-20080122/

Gonzalez, R., Jiang, L., Ahmed, M., Marciel, M., Cuevas, R., Metwalley, H. and Niccolini, S., 2017, June. The cookie recipe: Untangling the use of cookies in the wild. In *Network Traffic Measurement and Analysis Conference (TMA)*, 2017 (pp. 1-9). IEEE.

Google Cloud Platform, 2017, SQL Reference https://cloud.google.com/bigquery/docs/reference/standard-sql/

Grigorik, I., 2013, *HTTP Archive + BigQuery = Web Performance Answers*, in author's blog, https://www.igvita.com/2013/06/20/http-archive-bigquery-web-performance-answers/

International Organization for Standardization IEC JTC 1/SC 32, 2011, *ISO/IEC 9075-11:201, Information technology -- Database languages -- SQL -- Part 11: Information and Definition Schemas*, (SQL/Schemata). https://www.iso.org/standard/53685.html

Kamkar, S., 2010. Evercookie. *URL: http://samy.pl/evercookie*

Kimak, S., 2016. An investigation into possible attacks on HTML5 indexedDB and their prevention (Doctoral dissertation, Northumbria University).

Kontaxis, G. and Chew, M., 2015. Tracking protection in Firefox for privacy and performance. arXiv preprint arXiv:1506.04104. https://arxiv.org/pdf/1506.04104

Krishnamurthy, B. and Wills, C., 2009. *Privacy diffusion on the web: a longitudinal perspective*. In Proceedings of the 18th international conference on World wide web (pp. 541-550). ACM.

Kristol, D. and Montulli, L., 2009. *RFC 2965: HTTP State Management Mechanism*, 2000.

Kristol, D.M., 2001. HTTP Cookies: Standards, privacy, and politics. ACM Transactions on Internet Technology (TOIT), 1(2), pp.151-198.

Jayaraman, K., Du, W., Rajagopalan, B. and Chapin, S.J., 2010, June. *Escudo: A fine-grained protection model for web browsers*. In Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on (pp. 231-240). IEEE.

Jemel, M. and Serhrouchni, A., 2014, December. *Security enhancement of HTML5 local data storage*. In Network of the Future (NOF), 2014 International Conference and Workshop on the (pp. 1-2). IEEE.

Jobs, S., 2010. Thoughts on Flash. apple.com, April, 2010. URL: http://www. apple. com/hotnews/thoughts-on-flash/

Lekies, S. and Johns, M., 2012, November. *Lightweight integrity protection for web storage-driven content caching*. In 6th Workshop on Web (Vol. 2).

Mayer, J., 2011. Tracking the trackers: Self-help tools. The Center for Internet & Society. http://cyberlaw.stanford.edu/blog/2011/09/tracking-trackers-self-help-tools

Manico, J., 2009, *Real world cookie length limits*, in Manicode, http://manicode.blogspot.hk/2009/08/real-world-cookie-length-limits.html

McIlroy, M.D. and Kernighan, B.W., 1979. Unix Programmer's Manual. Vancouver

Mehta, N. R., 2009, *WebSimpleDB, A.P.I.*, in W3C Working Draft. https://www.w3.org/TR/2009/WD-WebSimpleDB-20090929/

Mehta, N., 2010, Indexed Database API, W3C Working Draft 05 January 2010 https://www.w3.org/TR/2010/WD-IndexedDB-20100105/

Microsoft, 2017, What is Silverlight? https://www.microsoft.com/silverlight/what-is-silverlight/default

Microsoft Developer Network, 2011, Introduction to Persistence, https://msdn.microsoft.com/en-us/library/ms533007(v=vs.85).aspx

Montulli, L., 1995, *Persistent client state in a hypertext transfer protocol based client-server system*, https://worldwide.espacenet.com/publicationDetails/biblio?CC=US&NR=5774670&KC=&FT=E&locale=en_EP

Mozilla, 2010, IndexedDB Security Review, https://wiki.mozilla.org/Security/Reviews/Firefox4/IndexedDB_Security_Review

Mylonas, A., Tsalis, N. and Gritzalis, D., 2013, September. Evaluating the manageability of web browsers controls. In *International Workshop on Security and Trust Management* (pp. 82-98). Springer, Berlin, Heidelberg.

Nisioti, A., Heydari, M., Mylonas, A., Katos, V. and Tafreshi, V.H.F., 2017, May. *TRAWL: Protection against rogue sites for the masses*. In Research Challenges in Information Science (RCIS), 2017 11th International Conference on (pp. 120-127). IEEE.

Nithyanand, R., Khattak, S., Javed, M., Vallina-Rodriguez, N., Falahrastegar, M., Powles, J.E., De Cristofaro, E., Haddadi, H. and Murdoch, S.J., 2016. *Adblocking and Counter Blocking: A Slice of the Arms Race*. In *FOCI*.

Oracle, 2013, Oracle Security Alert for CVE-2013-0422 in *Oracle Technology Network*, https://www.oracle.com/technetwork/topics/security/alert-cve-2013-0422-1896849.html

Oracle, 2017, Java Documentation, http://docs.oracle.com/en/java/

Olejnik, L., Minh-Dung, T. and Castelluccia, C., 2013. Selling off privacy at auction. <hal-00915249> https://hal.inria.fr/hal-00915249

Osmani, A., Cohen, M., 2017, Offline Storage for Progressive Web Apps https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa

Owens, M., 2006. Introducing SQLite. *The Definitive Guide to SQLite*, pp.1-16.

Quidsup, 2017, *NoTrack*, https://github.com/quidsup/notrack

Privacy Badger, 2017, https://www.eff.org/privacybadger

Preuveneers, D., Berbers, Y., Joosen, W., 2013. *The future of mobile e-health application development: exploring HTML5 for context-aware diabetes monitoring*. Procedia Computer Science, 21, pp.351-359.

Ranganathan, A., 2010. Beyond HTML5: Database apis and the road to indexdb. https://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb/

Ranganathan, A., Wilsher, S., 2010, Firefox 4: An early walk-through of IndexedDB, https://hacks.mozilla.org/2010/06/comparing-indexeddb-and-webdatabase/

Roberts, I., 2013 Browser Cookie Limits , http://browsercookielimits.squawky.net/

Roesner, F., Kohno, T. and Wetherall, D., 2012, April. *Detecting and defending against third-party tracking on the web*. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (pp. 12-12). USENIX Association.

Sendiang, M., Polii, A. and Mappadang, J., 2016, November. Minimization of SQL injection in scheduling application development. In *Knowledge Creation and Intelligent Computing (KCIC), International Conference on* (pp. 14-20). IEEE.

Shah, S., 2012, *HTML5 Top 10 Threats Stealth Attacks and Silent Exploits*. BlackHat Europe. http://www.hakim.ws/BHUSA12/materials/Briefings/Shah/BH_US_12_Shah_Silent_Exploits_WP.pdf

Schwartz, A., Sherry, L., Cooper, M., Tien, L., Pierce, D., Brandt, D., President, Smith, R. E., Givens, B., Dixon, P., 2007, Consumer Rights and Protections in the Behavioral Advertising Sector, https://www.cdt.org/files/privacy/20071031consumerprotectionsbehavioral.pdf

Shepherd, E., 2017, *Same-origin policy* in MDN Web Docs https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

Soghoian, C., 2011. *The history of the do not track header*. Slight Paranoia. Retrieved 28 October 2013 from http://paranoia. dubfire. net/2011/01/history-of-donot-track-header.html

Soltani, A., Canty, S., Mayo, Q., Thomas, L. and Hoofnagle, C.J., 2010, March. *Flash Cookies and Privacy*. In AAAI spring symposium: intelligent information privacy management (Vol. 2010, pp. 158-163).

Sounders, S., 2011, *Announcing the HTTP Archive*, High performance web sites blog, https://www.stevesouders.com/blog/2011/03/30/announcing-the-http-archive/

Tsalis, N., Mylonas, A., Nisioti, A., Gritzalis, D. and Katos, V., 2017. *Exploring the protection of private browsing in desktop browsers*. Computers & Security, 67, pp.181-197.

Tump, E., 2011, *The Risks of Client-Side Data Storage*, Sans Institute, https://www.sans.org/reading-room/whitepapers/dlp/risks-client-side-data-storage-33669

Virvilis, N., Mylonas, A., Tsalis, N. and Gritzalis, D., 2015. *Security Busters: Web browser security vs. rogue sites*. Computers & Security, 52, pp.90-105.

Web Hypertext Application Technology Working Group, 2017, Web storage in HTML Living Standard. https://html.spec.whatwg.org/multipage/webstorage.html

W3C, 2012, *HTTP Archive (HAR) format*, https://w3c.github.io/web-performance/specs/HAR/Overview.html

West, W. and Pulimood, S.M., 2012. Analysis of privacy and security in HTML5 web storage. *Journal of Computing Sciences in Colleges*, *27*(3), pp.80-87.

Y. Wu, D. Meng and H. Chen, 2017, Evaluating private modes in desktop and mobile browsers and their resistance to fingerprinting, in IEEE Conference on Communications and Network Security (CNS), Las Vegas, NV, USA, 2017, pp. 1-9.

8. APPENDICES

8.1. APPENDIX D:

8.1.1. Analysing the HTTP Archive dataset with SQL

This section illustrates a set of SQL queries that can be used to verify the presence of code constructs that relate to certain primitives in the HTTP Archive dataset. The example given below employs the matching rules defined for Web SQL Database. In this work, similar queries were used to verify the presence of the other primitives considered. For each one of them, the content of the regular expressions was adjusted in order to reflect the relevant matching rules.

The query presented below returns a table containing statistical data about the dataset considered. After verifying the occurrence of certain constructs, it matches the hostnames of the resulting subresources against a database of known trackers. The latter was generated using 'blacklist-merger', a tool, also described in Chapter 5, that aggregates data from a selection of curated blacklists of user trackers. Moreover, the query also returns distinct statistical data regarding third-party subresources. It also returns the percentage of truncated bodies in the dataset. This value can be considered as the margin of error of this methodology, because it reflects the amount of subresources that cannot be fully inspected.

```
WITH
  matches AS (
  SELECT
    page,
    url
  FROM
    `httparchive.har.2017_09_15_chrome_requests_bodies`
  WHERE
    REGEXP_CONTAINS (body, r'openDatabase')
    AND REGEXP_CONTAINS(body, r'transaction')
    AND REGEXP_CONTAINS(body, r'executeSql')),
  third_party_matches AS (
  SELECT
    page,
    url
  FROM
    `httparchive.har.2017_09_15_chrome_requests_bodies`
  WHERE
    REGEXP_CONTAINS (body, r'openDatabase')
    AND REGEXP_CONTAINS(body, r'transaction')
    AND REGEXP_CONTAINS(body, r'executeSql')
    AND NET.REG_DOMAIN(page) != NET.REG_DOMAIN(url)),
  total AS (
  SELECT
```

```
    page,
    url,
    truncated
  FROM
    `httparchive.har.2017_09_15_chrome_requests_bodies` )
SELECT
  *,
  /*percentages sites
  */ websites_with_construct_in_subresources / websites_considered * 100 AS
percentage_of_websites_with_construct_in_subresource,
  websites_with_construct_in_trd_party_subresources / websites_considered *
100 AS percentage_of_websites_with_construct_in_trd_party_subresources,
  sites_with_blacklisted_subresources_with_construct / websites_considered
* 100 AS percentage_of_sites_with_blacklisted_subresources_with_construct,
  sites_with_blacklisted_third_party_subresources_containing_construct /
websites_considered * 100 AS
percentage_of_sites_with_blacklisted_third_party_subresources_containing_co
nstruct,
  /*percentages sub
  */ subresources_with_construct / subresources_analysed * 100 AS
percentage_of_sub_containing_construct,
  trd_party_subresources_with_construct / subresources_analysed * 100 AS
percentage_of_trd_party_sub_containing_construct,
  blacklisted_subresources_with_construct / subresources_with_construct *
100 AS blacklisted_perc,
  blacklisted_third_party_subresources_containing_construct /
subresources_with_construct * 100 AS
perc_blacklisted_third_party_subresources_containing_construct,
  blacklisted_subresources_with_construct / subresources_analysed * 100 AS
blacklisted_perc_total,
  /*margin of error
  */ truncated / subresources_analysed * 100 AS trunc_perc
FROM (
  SELECT
    /*
    1. Counts all website in the scrap
    */(
    SELECT
      COUNT(DISTINCT page)
    FROM
      total) AS websites_considered,
    /*
    2. Counts all subresouces in the scrap
    */(
    SELECT
```

```
      COUNT(url)
    FROM
      total) AS subresources_analysed,
    /*
    3. Counts all websites with at least one src with the construct
    */(
    SELECT
      COUNT(DISTINCT page)
    FROM
      matches) AS websites_with_construct_in_subresources,
    /*
    4. Counts all websites with at least one 3rd party src with the
construct
    */(
    SELECT
      COUNT(DISTINCT page)
    FROM
      third_party_matches) AS
websites_with_construct_in_trd_party_subresources,
    /*
    5. Counts all subresources with the construct
    */ (
    SELECT
      COUNT(url)
    FROM
      matches) AS subresources_with_construct,
    /*
    6. Counts all 3rd party subresources with the construct
    */ (
    SELECT
      COUNT(url)
    FROM
      third_party_matches) AS trd_party_subresources_with_construct,
    /*
    7. Counts all trucated bodies
    */(
    SELECT
      SUM(CAST( truncated AS INT64)) AS number_of_truncated_bodies
    FROM
      total) AS truncated,
    /*
    8. Counts all blacklisted subresources with the construct
```

```
    */(
    SELECT
      COUNT(url) AS number
    FROM
      matches
    INNER JOIN
      `pacta-96bd2.TrackersList.notrack` AS blacklist
    ON
      NET.REG_DOMAIN(matches.url) = blacklist.domain) AS
blacklisted_subresources_with_construct,
    /*
    9. Counts all websites with blacklisted subresources with the construct
    */ (
    SELECT
      COUNT(DISTINCT page) AS number
    FROM
      matches
    INNER JOIN
      `pacta-96bd2.TrackersList.notrack` AS blacklist
    ON
      NET.REG_DOMAIN(matches.url) = blacklist.domain) AS
sites_with_blacklisted_subresources_with_construct,
    /*
    10. Counts all 3rd party blacklisted subresources containing the
construct
    */ (
    SELECT
      COUNT(url) AS number
    FROM
      third_party_matches
    INNER JOIN
      `pacta-96bd2.TrackersList.notrack` AS blacklist
    ON
      NET.REG_DOMAIN(third_party_matches.url) = blacklist.domain) AS
blacklisted_third_party_subresources_containing_construct,
    /*
    11. Counts all websites with 3rd party blacklisted subresources
containing the construct
    */(
    SELECT
      COUNT(DISTINCT page) AS number
    FROM
      third_party_matches
```

```
    INNER JOIN
      `pacta-96bd2.TrackersList.notrack` AS blacklist
    ON
      NET.REG_DOMAIN(third_party_matches.url) = blacklist.domain) AS
sites_with_blacklisted_third_party_subresources_containing_construct)
```

8.1.2. Targeting the analysis to the Alexa Top 10000 sites

The queries presented below can be used to analyse a subset of the HTTP Archive dataset that only contains data relating to websites that score 10000 or above in the Alexa ranking. Since the ranking is not included in the main dataset used by the previous query, it is necessary to retrieve it from another table offered by the HTTP Archive project. This can be done by first creating a table that lists the domain names of sites whose ranking is 10000 or higher.

```
SELECT
  *
FROM
  `httparchive.runs.2017_09_15_pages`
WHERE
  rank < 10001
```

The following step entails using the table created above to filter the whole dataset, by performing an inner join operation on the domain name.

```
SELECT
  full_list.*
FROM
  `httparchive.har.2017_09_15_chrome_requests_bodies` AS full_list
INNER JOIN
  `pacta-96bd2.September.15_top_10000_pages` AS top10k
ON
  full_list.page = top10k.url
```

The query outputs a new table that follows the same schema of the main HTTP Archive dataset and can, therefore, be passed to the query presented on paragraph 9.4.1. As mentioned in Chapter 5, it is important to make sure that the dates of the datasets used are consistent. In this example, for instance, it is necessary verify that the Alexa ranking values used reflects the actual ranking of the sites at the point in time in which the HTTP Archive scan was performed.