Insert here your thesis' task.

Czech Technical University in Prague
Faculty of Information Technology
Department of Computer Systems

Master's thesis

Website user tracking

Bc. Adam Prášil

Supervisor: Ing. Tomáš Zahradnický, Ph.D.

11th January 2016

Acknowledgements

I would like to express my gratitude and appreciation to my supervisor Dr. Zahradnický for his guidance.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 11th January 2016 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Prášil, Adam. *Website user tracking.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Hodnota informací o aktivitách uživatelů na internetu v poslední době stále roste a některé internetové stránky sbírají o uživatelích tolik informací, kolik je jen možné, často bez ohledu na soukromí uživatele. Tato práce analyzuje metody, kterými si mohou webové stránky pomocí standardních technologií označit či zapamatovat své návštěvníky a později je identifikovat. Pomocí referenční implementace ukazuje sílu těchto metod a zranitelnost běžných prohlížečů vůči nim.

Klíčová slova soukromí na internetu, sledování uživatelů, otisk prohlížeče, cookie, supercookie

Abstract

The value of information about the user activity on the Internet is rising today and some websites collect as much information as it is possible, no matter of user's privacy. This thesis analyzes various methods using standard technologies to mark or remember website visitors and recognize them later. It shows these methods' power in a reference implementation, which is used to test common browsers' vulnerability to these methods.

Keywords Internet privacy, user tracking, browser fingerprint, cookie, supercookie

Contents

List of Figures

List of Tables

Chapter 1

Introduction

On 10th December 1948, the General Assembly of the United Nations adopted and proclaimed the Universal Declaration of Human Rights, which in the 12th article states that *no one shall be subjected to arbitrary interference with his privacy, family, home or correspondence, nor to attacks upon his honour and reputation. Everyone has the right to the protection of the law against such interference or attacks* [1]. The document, including its 12th article, is reflected in the law of many countries around the world and in the Charter of Fundamental Rights and Freedoms, a part of the constitutional order of the Czech Republic, the right to privacy is reflected in the 7th, 10th, and 13th article.

The user's privacy on the Internet can be threatened by websites using various tracking techniques. These techniques allow a website to mark or remember a user and recognize her in subsequent actions. No matter if the tracking is used within a single website or across more sites, both lead to a loss of privacy.

The cookie is the basic technique to track the user's activity on the Internet. Its initial purpose was to enable websites to store user specific data, like a session identifier, on the client's side, but it can be abused to store an identifier for tracking purposes.

Because the cookie became well known and could be easily detected or blocked, new ways to track the user were invented. When a website stores a user identifier somewhere else than in the browser cookie storage, it is called a "supercookie". This technique usually abuses available browser features to store a persistent piece of information into user's computer.

Another approach to user tracking is to retrieve information accessible to the website and use it to create a unique fingerprint of the user's device. The fingerprint is then stored on the server and used to identify the user in the future.

No matter whether a website performs tracking using cookies, supercookies, or fingerprinting methods, it is an attack on user's privacy. The thesis

will describe tracking methods in detail, showing how they work on a reference implementation and assessing their potential by testing on common web browsers.

The thesis is organized as follows. After introducing the reader into the topic of the thesis, chapter 2 presents user tracking history and its common aspects. Chapter 3 presents an analysis of tracking techniques, divided into three sections – cookies, supercookies, and fingerprinting. Chapter 4 presents details about the implemented showcase of tracking mechanisms, including results of methods testing on common web browsers, while chapter 5 concludes the thesis.

Chapter 2

State of the Art

Because the HTTP protocol is designed as a stateless protocol [2], there was no way to mark the user and track her activity in the beginning of the World Wide Web. The only option was to send a parameter – usually GET – between each web request. When the user closed her browser, everything got forgotten and her next visit of that page looked like she visited the page for the first time. The user privacy was protected but the websites possibilities were very limited and it was difficult to develop any richer application.

The change came back in 1994 when Lou Montulli presented an idea of storing website state in clients' devices [3]. This concept is well known as "cookies". Microsoft implemented it in Internet Explorer 2.0 [4] in November 1995 [5] and Netscape Navigator – the most used website browser at that time [6] – supported cookies from the first version [7]. The whole concept was described for the first time in RFC 2109 [8] in 1997.

Cookies are the basic concept for websites to store the user's state – they are usually used to store a session identifier for the logged-in user but technically, they can be used to store any string value. For safety reasons, all data is usually stored on the server, while on the client side only a generated identifier [9] is saved, even if the user is not logged-in. Cookies are well known to common users, because many websites contain a disclaimer that they are using cookies and cookies are also especially mentioned in some browsers.

Caching of the static web content is also available from the beginnings of the World Wide Web. The browser stores the content returned from the server locally and the caching mechanism can be easily abused to store a unique identifier, as it is described in the section 3.2.1. The identifier can be stored simple in the cached content body or alternatively in cached content metadata. RFC 2068 [10] in 1997 defined the ETag header, which serves as a cached content version tag and can be abused for tracking [11] in a way described in the section 3.2.2.

As the demand of more interactive websites increased, a scripting language was created to manipulate the Document Object Model (DOM) [12] and to

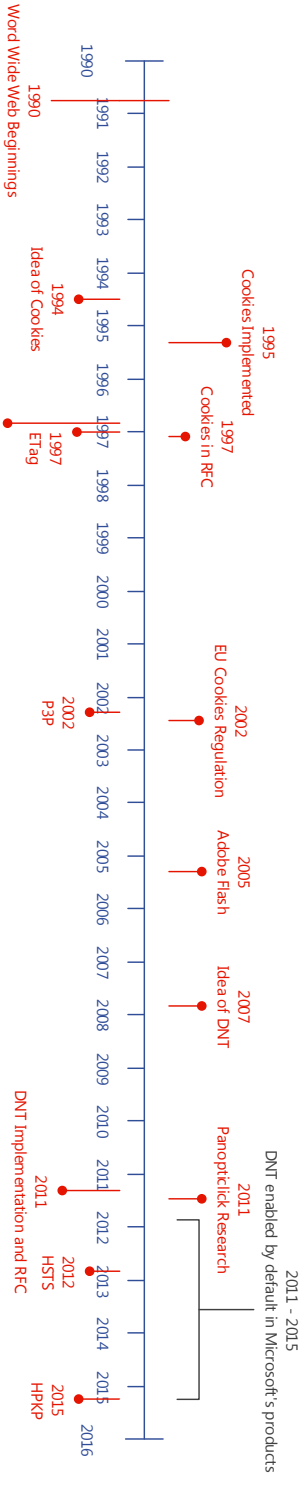

Figure 2.1: Timeline of significant events regarding user tracking

make the website more interactive. In the beginning, each browser developed its own language, but in 1997 was created the first ECMA standard [13] and its implementation JavaScript, trying to unite web browser scripting languages. Nowadays most functionality works the same among the web browsers, though minor differences persist.

In 1996 the Macromedia Flash technology [14] was introduced. It was acquired by Adobe Systems Incorporated in 2005 and it is now known as Adobe Flash. It provides an option for websites to contain interactive multimedia content like animations, games, and videos – Flash was an important niche filler especially in the beginning of the World Wide Web. Nowadays Flash is usually talked about because of many security issues [15] and it also provides the storage which can be abused for tracking [16], as described in the section 3.2.5.

As the World Wide Web grew larger, new questions about privacy of the users raised. One of the first attempts to deal with tracking using cookies was the Platform for Privacy Preferences Project (P3P), which was presented for the first time in 2002 as the W3C P3P 1.0 Specification [17]. The idea was that websites should inform the browser about the purpose of their cookies and the browser should decide whether to allow the website to store the cookie or not. However, only Internet Explorer and Edge browsers now support it [18] and even Microsoft marked it as deprecated. The main reason for that probably is that many websites did not implement P3P right or did not implement it at all [19].

The European Union legislation contains cookies regulation [20]. The first attempt to regulate cookies was made in the Regulation (EC) No 45/2001 [21], followed by the Directive 2002/58/EC of the European Parliament and of the Council of 12 July 2002 [22], stating that if *cookies are intended for a legitimate purpose, such as to facilitate the provision of information society services, their use should be allowed on condition that users are provided with clear and precise information in accordance with Directive 95/46/EC about the purposes of cookies or similar devices so as to ensure that users are made aware of information being placed on the terminal equipment they are using. Users should have the opportunity to refuse to have a cookie or similar device stored on their terminal equipment.* This requires users be informed about using cookies on websites although it contains an exemption for common regular use. Analytics or advertisement usage is explicitly mentioned as not exempt [23]. As already stated, cookies usually contain just a simple string identifier and users never can be sure what is happening with collected data on the server site.

In 2007 a discussion [24] about a simple mechanism called Do Not Track (DNT) started. This mechanism was intended to enable users to set their preference and the browser to inform the website that the user does not want to be tracked by cookies or similar methods. Though there were more variants, it ended with a simple `DNT` header [25] in the HTTP request. If it is set to

the value 1, the user prefers not to be tracked on the website. The final solution was presented by Mozilla in 2011 [26] and others followed soon. Nowadays, all modern browsers contain an option to control whether to send the `DNT` header or not, but it is usually turned off by default.

However, in 2012 Microsoft released Internet Explorer 10 which had the Do Not Track option enabled by default [27]. Therefore a common user, who does not ever heard about the `DNT` header, sent `DNT:1` to all the servers she visited, just because she clicked on the express settings option when she launched the browser for the first time. This choice lead to protests of the advertisement companies [28] and since Microsoft did not change their policy, some web servers started to ignore this header [29]. Microsoft kept this setting as default in Internet Explorer 11, released in 2013, but in 2015 Microsoft finally changed their approach to Do Not Track [30] and DNT is not enabled by default in the express settings anymore.

In 2011 the Electronic Frontier Foundation (EFF) in their research project called Panopticlick [31] proved that there is another way to track users – more hidden and hard to detect or defend against – fingerprinting. It is possible, because modern web browsers provide many functions [32] revealing information about the user's device. If they are combined together, the result can serve as a unique or almost unique user identifier – fingerprint. According to the Panopticlick research, fingerprints contain at least 18.1 bits of entropy and it is just an expected lower bound. In fact, the real entropy is probably higher because of additional information retrievable from the browser which was not included in their tests, for instance data obtained from ActiveX plugins or by CSS font detection [33].

The problem with user tracking by fingerprinting is that many attributes used for fingerprinting can change over time so when a user returns to the site, the former fingerprint does not need to match the stored one. The EFF group dealt with this problem in its research by simple algorithmic filtering of specific cases and using the sequence matcher from the Python Standard Library [34] for counting similarity of two strings. They checked their predictions by the user identifier stored in a common HTTP cookie and they figured out that the algorithm made a correct guess in 99.1 % of cases [31].

There is initiative in the last years to solve the insecure HTTP connections made between the browser and the server. For that reason started the initiatives like Let's Encrypt [35] to provide a valid certificate issued by a trustworthy certification authority [36] to most websites. The HTTPS connection, however, still can be attacked by the man-in-the-middle (MITM) attack [37]. The HTTP Strict Transport Security (HSTS) [38] was standardized in 2012 and its purpose was to solve the problem of the SSL Strip [39], a special kind of MITM attack. In 2015 was standardized an another mechanism called HTTP Public Key Pinning (HPKP) [40] to allow websites to pin their certificate in the browser and defend against rogue certificate authorities [41] and against vulnerabilities of certificate authorities [42] or DNS [43] providers. Although

they were designed to protect the users, both can be abused to track the user's activity on the Internet [44]. How could it be done is described in sections 3.2.3 and 3.2.4.

The current tracking methods can be divided into the following three categories:

1. **Cookies (section 3.1)** Simple HTTP cookies as defined in RFC [45].
2. **Supercookies (section 3.2)** Methods used to store an identifier in the user's device, other than the regular HTTP cookie.
3. **Fingerprinting (section 3.3)** Methods used to retrieve as much information from the device as possible to uniquely identify the same device later.

These categories will be used to structure the analysis part of the thesis.

A frequent reason to mark or remember a user is storing a web application's state for the current user – usually called a session. In that case, an HTTP cookie is sufficient, because there is no expectation that the user would block or delete it. There also exists an alternative way – sending the session identifier as a GET/POST parameter in each request. There is no reason to use the advanced user tracking method like supercookies or fingerprinting for this purpose.

Advanced tracking methods are useful when a webpage needs to track activity of the user which is currently not logged in. Common cookies can still be used but when there is a need to have correct statistics without any interference from the plugins like AdBlock [46] or 3rd party software like antivirus or privacy suites, or to track the user inconspicuously, supercookies and fingerprinting can become useful.

It may look harmless when a company uses some advanced techniques to track the user's activity on their own page – common use case is when user returns to a website of an e-shop and there pops up an advertisement with the product she has explored last time. The problem arises when many websites contain resources from the same URL, which means for instance advertisement services, share buttons, or analytics. Providers of these kinds of services can abuse [47, 48] the described methods to collect a substantial amount of information about specific user's activity on the Internet.

We have summarized the history of the website user tracking from the World Wide Web beginnings up to the present day. There are three categories of tracking methods – cookies, supercookies, and fingerprinting. These methods will be analyzed in the next chapter.

Chapter 3

Analysis

As already mentioned, the current HTTP standard includes a mechanism to store a user's session state on the user's device – cookies (3.1). Technically, the same mechanism can be used for user tracking but it has few disadvantages nowadays – the user can easily delete cookies in all modern browsers, they do not work in the anonymous[1] web browsing mode of the browser, tracking is easily detectable by any visitor, various privacy or security tools block cookies, and they are also covered by the law.

The user can be tracked by more sophisticated, better working, worse detectable, and more covert methods which can be divided into two categories – supercookies (3.2) and fingerprinting (3.3). The main difference between the two categories is that if the server uses supercookies, it tries to hide a generated identifier somewhere in the client's device, while in case of fingerprinting the server tries to retrieve enough unique information from the client, saving it on the server side.

In this chapter we will present known methods from both categories, first the initial purpose and technology description, followed by abusing opportunities. We will examine strengths and weaknesses of the described methods.

3.1 Cookies

Cookies are the basic concept of storing website data in the user's device. They are supported in all modern browsers and they are also well known to the users. The whole concept is called the HTTP State Management Mechanism in RFCs and it was described for the first time in RFC 2109 [8], published in 1997. The definition has been obsoleted in 2000 by RFC 2965 [49], which requires using quotes for the cookie values, presents new features like filtering

[1] The anonymous browser mode is called the "private mode" or "incognito mode" in some web browsers.

based on a request port, and contains new cookie attributes, like `CommentURL` or `Discard`.

RFC 2109 presented two new HTTP headers – `Set-Cookie` and `Cookie`. The first header is sent by the server to the client as a request to store a cookie locally at the client's device, while the second one is used by the client to send the stored cookie back to the server along with an HTTP request. In order to avoid compatibility problems RFC 2965 presented new headers `Set-Cookie2` and `Cookie2`. Using different header names was intended to make the client and the server sure that they both use the same version of the standard.

However, in 2011 was published RFC 6265 [45], which obsoletes RFC 2965 from 2000 and it is the current standard for the cookies implementation. It contains some changes in the cookie attributes and redefines how cookies should be accessible across subdomains of the same domain. The new definition uses the `Set-Cookie` and `Cookie` headers, the same ones as the obsolete RFC 2109. According to RFC 6265, if the server sends the `Set-Cookie` header, it should mean that it implements cookies the way described in RFC 6265. As opposite, if the server sends the `Set-Cookie2` header, it should implement cookies as it is described in RFC 2965. However, there is no guarantee that the real web browsers and servers implement cookies mechanism the way stated in either RFC.

When a website uses the cookie to track a user, it just generates some identifier when the user visits it for the first time and returns the identifier along with the first HTTP response in the `Set-Cookie` header. The browser automatically stores the retrieved cookie in the browser cookie storage and adds it as the `Cookie` header in each subsequent request to the same domain. The whole process is illustrated at Fig. 3.1.

3.2 Supercookies

There are web technologies other than cookies that have to store data somewhere in the user's device. Some of them, like Web Storage [50], are based on storing data in browser, others need to store data to work properly – for instance HSTS [38] or ETag [51]. Although these methods purpose is not to store the identification data about the user, they can be abused to do so [47]. There is not any precise and commonly accepted definition of a supercookie, but for the purpose of this thesis it can be defined as **a cookie stored elsewhere than in the browser cookie storage**. The supercookie is sometimes [47] called an "evercookie", by the Evercookie project [52].

Supercookies can be used as an additional tracking mechanism to cookies or alone as a worse detectable and more resilient user tracking method. They can also be used for a cookie backup so when an HTTP cookie is deleted, it is restored from the supercookie in the next request. This method is sometimes called a "zombie cookie" [53].

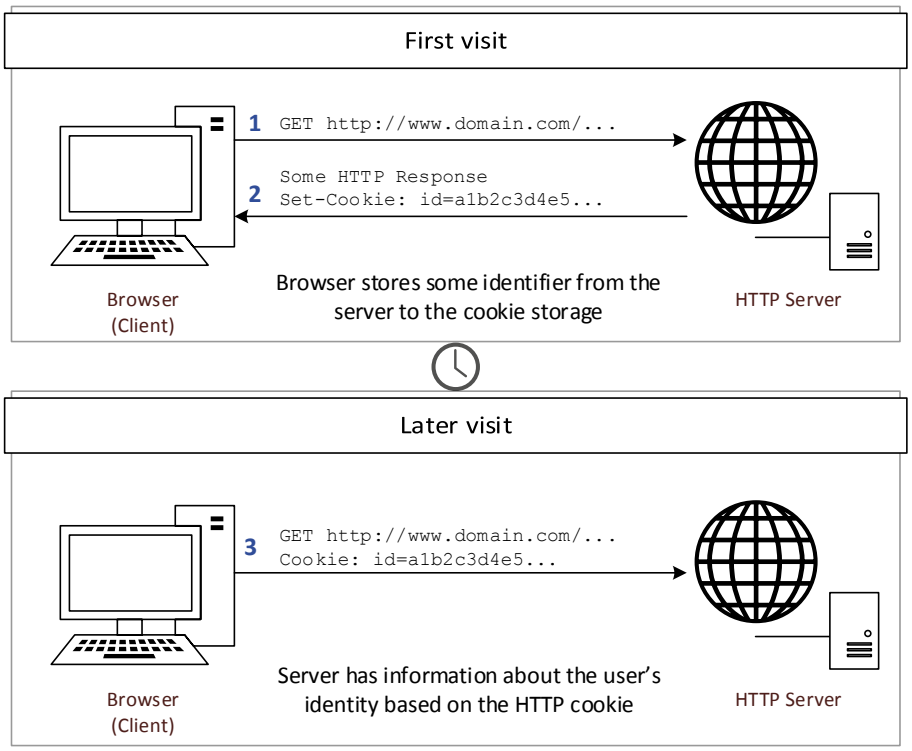

Figure 3.1: Cookie tracking mechanism

The first time a user visits a website (1), the `Set-Cookie` header is returned (2), instructing the browser to store a generated identifier. The identifier is then sent along with the later request (3), telling the server that the request is performed by the same user.

Deleting regular HTTP cookies in the browser usually does not affect supercookies. Therefore, to get rid of the supercookies, the user has to wipe all data from her profile or use a third party tool to delete known supercookies. However, we will see that there are many kinds of supercookies and it is difficult to delete them all.

3.2.1 Cached Resources

Websites on the Internet usually contain more than just HTML tags and text content. According to stats gathered by the HTTP Archive on 15 th September 2015 [54], size of an average website on the Internet was 2 182 KiB. Fig. 3.2 shows how the average size is distributed by the Content Type.

It can be observed that HTML code takes 56 KiB out of 2 182 KiB meaning that the remaining 97.43 % of an average website is occupied by static files that change infrequently, for instance when a new version of the web application is released. If the browser was downloading all the website content each time

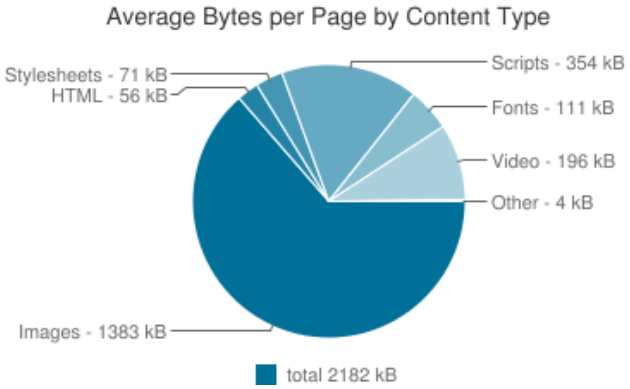

Figure 3.2: Average page size by the Content Type on 15th September 2015 [54]

when the user visited it, or just clicked on the internal link on that page, there would probably be in average 97.43 % of useless data traffic in each request. It is the reason why browsers usually cache the static content.

The simplest way to save a supercookie in user's device is just to put it into the cached content. Doing so does not need any special browser features, it just needs caching enabled. This approach is supported in most used browsers across all platforms, including old versions of browsers. The method can be perfectly used for cookie backup. The cookie is stored traditionally – using the `Set-Cookie` header – in the browser cookie storage and its backup is placed to the cached content, for instance to an embedded JavaScript file, as a static variable. The file is cached by the browser and when the user returns to the website, the file is loaded from the browser cache and can be used to restore the deleted or modified cookie. Moreover, using the cached resource supercookie enables websites to track the user across the domains by including the file from the same URL in each website. The described principle using JavaScript is demonstrated at Fig. 3.3.

When the website saves a supercookie into the cached content, it has to be processed through some application logic on the server to provide a unique version of the requested resource for each user. For that reason, the resource cannot be served by a simple HTTP server, it has to be served by some application server capable of generating unique versions of the served resource.

User tracking using a cached JavaScript resource is the simplest way how to use cache content to store supercookies. Each user obtains from the server a unique JavaScript file containing a static variable with a user identifier. The variable can be used to restore the cookie in the same or in the different JavaScript file. However, the user identifier can be stored in a cached image as well, reading it back from pixel RGB values using HTML5 canvas element [55] and its JavaScript functions. The identifier can be either stored in a visible

Figure 3.3: Process of storing a supercookie in the cached JavaScript resource

The first time a user visits a website (1), a regular cookie is set in a response (2). Then the browser automatically loads an embedded JavaScript file (3), which is stored in the cache for an amount of time defined in a response (4). The cached JavaScript contains a variable with a cookie backup and a function restoring the cookie value (5). Therefore when the regular cookie is deleted, it is restored from the backup in the cached JavaScript.

image, using steganography [56], or in a hidden image, using all possible pixel values to store the identifier. Supercookies stored in the cached images are well hidden but it is more difficult to generate these files and to retrieve the value back in JavaScript.

Technically, CSS files can be also used to store supercookies, setting specific style attributes to DOM objects [57]. Most of CSS attributes can be abused for this purpose when they are combined together, each of them containing a bit of an identifier. The most information can be stored in string attributes like `background-image`, in color attributes like `background-color`, or in groups of number attributes like `margin`. The identifier is later read by JavaScript creating the object in the document, having the specific class or id assigned. Browser automatically applies CSS styles to created object making them accessible through JavaScript methods.

User tracking using cached resources is easily detectable just by performing a request to the resource file URL from more computers. If each computer receives a distinct content of the requested resource file, this method is probably used. Disabling browser cache could work as a defense against this method, but it makes browsing websites much less comfortable. In 2011 came out [58] that Microsoft used this method for restoring cookies on their `live.com` domain.

3.2.2 HTTP ETag

The HTTP protocol contains a mechanism called the entity tag, shortly ETag, which is, by definition "*an opaque validator for differentiating between multiple representations of the same resource*" [51]. This mechanism is often used to avoid the useless network traffic giving the servers an option to inform clients about change in the resource content. When a client asks a server for a resource, the server can return a special header informing the client about version of the returned resource. Next time the client will send the version identification along with the request to the server, which can base on it the decision whether to send the full content of the resource or just the `NOT MODIFIED` status. The current version of this process is described in RFC 7232 [51] and shown at Fig. 3.4.

3.2.2.1 ETag Abuse

The main problem with this mechanism is that there are no restrictions on the version identifier – it can be any string, even its length is not restricted by definition. If this feature is enabled on a common HTTP Server like Apache, it usually returns the result of some hash function with defined inputs like file name, last modification date and file size. Applications can abuse this header to send a user identifier knowing that when the request for content with that ETag comes back to the server, it is originated from the user marked before.

As long as this method abuses browsers cache, users can protect themselves by disabling or periodically clearing cache in their browsers. This would work but it is not quite comfortable because the websites takes much time to load without cache. It can be also expensive in case of mobile connections with limited data traffic. Classic methods like disabling JavaScript and cookies will not work against ETag user tracking.

Using this kind of tracking methods is easily detectable by performing a number of requests – better from different computers – in short time for the same content which is not supposed to change. Results should be either the same or from a limited set which can mean badly configured load balancing where each server has different last modification time of the file or the ETag contains internal hostname of the server. If the returned ETag is different in every response the server probably abuses ETag to uniquely mark its clients. In 2011 came out [58] that Microsoft used this method for restoring cookies on their domain `live.com` along with the cached JavaScript resources.

3.2.3 HTTP Strict Transport Security

The first attempt to develop secure communication over the HTTP protocol was made in 1993 by Netscape Communications. In 1994 they came up with Secure Sockets Layer (SSL) 1.0, which was only internal version because of many security problems. This version was replaced by SSL 2.0 in the same

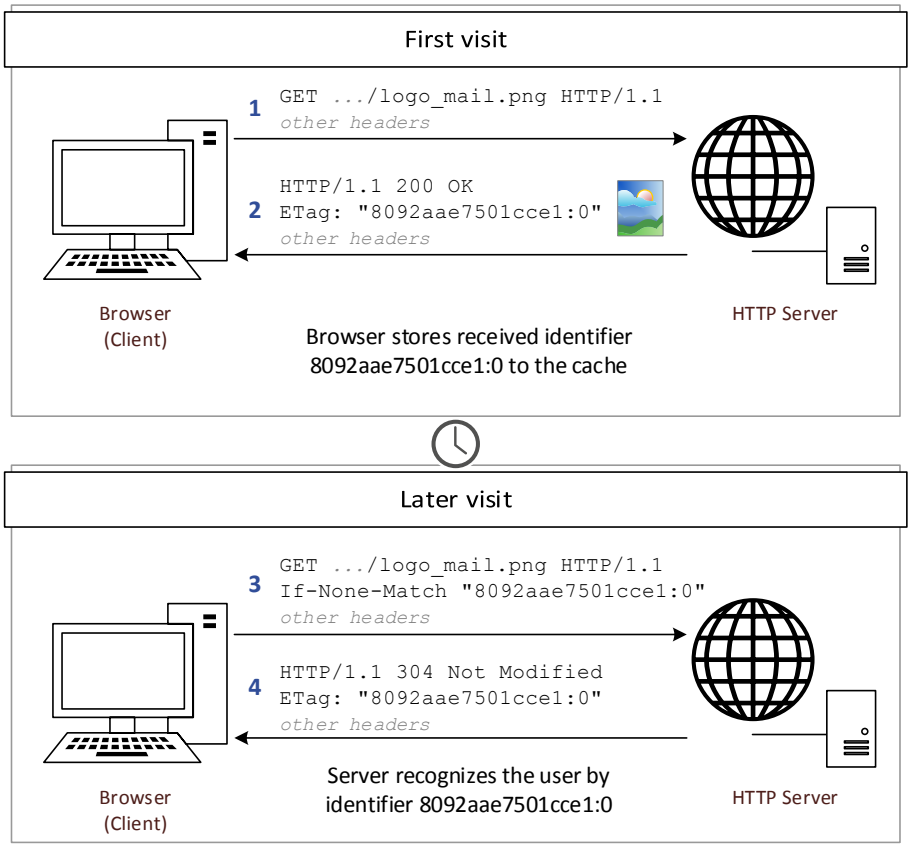

Figure 3.4: HTTP ETag tracking mechanism

The first time a user visits a website, an embedded static content request is performed (1). Then the server sends the `ETag` header along with the response (2). The browser stores the returned content and its entity tag into the cache. Next time the user visits the website, the embedded static content request is performed again, but this time the request contains the `If-None-Match` header with the stored entity tag as a value (3). Server returns the `Not Modified` status so the content in the cache is not refreshed (4), but server can identify client from the retrieved entity tag.

year and one year after, in 1995, was released SSL 3.0 – from this version has evolved the standardized protocol called Transport Layer Socket (TLS) which was improved by years and now it is used as a standard for secure communication over the HTTP protocol [36] – shortly HTTPS.

At Fig. 3.5 is evident, that most website requests on the Internet is still performed using the unencrypted HTTP. Despite initiatives like `StartSSL` [59] or `Let's Encrypt` [35], many domains still do not use a certificate issued by a trustworthy certification authority, therefore they support HTTPS with a self-signed certificate only or they do not support HTTPS at all.

If a user wants to access a website on domain `domain.com`, she usually types `www.domain.com` or just `domain.com`. Most users, however, omit either

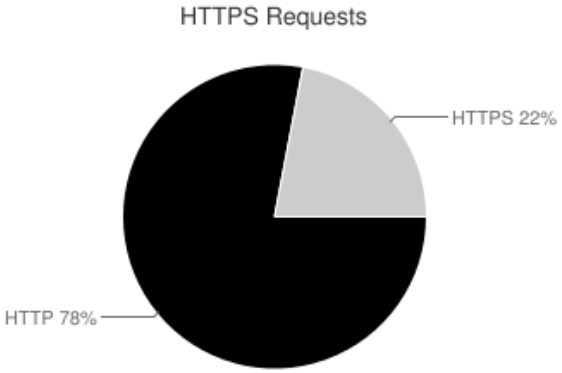

Figure 3.5: HTTPS requests percentage on 15 th September 2015 [54]

the `http://` or `https://` prefix in the address. After entering the incomplete address, the browser usually queries the website through the HTTP protocol, because it has no information about the HTTPS support on the server. In the response, the server can redirect the browser to the HTTPS protocol but the first request is not encrypted, therefore it is vulnerable to man-in-the-middle (MITM) attack. This kind of man-in-the-middle attack is called SSL Strip and was presented at Black Hat DC 2009 [39]. An example of the SSL Strip attack to HTTPS communication is described at Fig. 3.6.

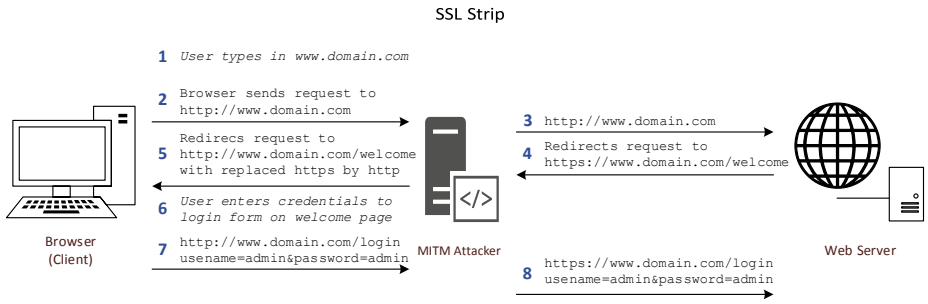

Figure 3.6: Sample SSL Strip attack stealing a user credentials by a man-in-the-middle attacker

When a user types in the address field `www.domain.com` (1) and the browser does not know anything about the target address, a request to `http://www.domain.com` is performed (2,3). If the server requires secure communication, a redirection to `https://...` address is returned (4). The attacker can catch the redirection and change it back to `http://...` along with changing all the URLs in the website from `https://...` to `http://...` (5), resending all requests from the client to `https://...` address of the server. Therefore the attacker can see the whole communication in plaintext (6) and it is not spotted, because from the browser's point of view it looks like the server does not support secured communication (7), while the server communicates "securely" (8) with the attacker.

As a protection against this kind of attack it was created a mechanism called HTTP Strict Transport Security (HSTS) which is described in RFC 6797 [38]. The mechanism allows the server to set a special header `Strict-Transport-Security` in the response, informing the client that the server supports HTTPS. The browser stores this information locally for a time length specified in the `max-age` property and the next request will be redirected to HTTPS internally by the browser, without calling remote server over HTTP first. This mechanism protects user from SSL Strip attack assuming that the first request is done through a secure link. There also exists a solution for securing the connection without relying on the first secure request – an HSTS preload list [60] stored in the web browser installation.

3.2.3.1 Strict Transport Security Header Abuse

This mechanism allows the server to inform the client that HTTPS is supported by the whole domain with all its subdomains or just by a concrete subdomain – using the flag `includeSubDomains`. Information about HTTPS support is stored in the browser for an amount of time, specified in the `max-age` parameter of the `Strict-Transport-Security` header. Thus when the company owns a wildcard certificate for its domain, it can store one bit of information in user's browser for each of its subdomain. The first time the user visits a page, an identifier is generated and stored as bits by returning the `Strict-Transport-Security` header for domains corresponding to the bits which should be true. On the next visits, a request is sent to each subdomain through `http` and a browser redirection to `https` is detected by JavaScript or on the server side – the identifier is then retrieved bit by bit, occurrence of each redirection provides a single bit of information. From the bits can be retrieved the previously stored identifier as it's illustrated at Fig. 3.7.

Technically, there can be stored many bits of information in the browser this way, but each bit requires its own request to the server. Nowadays websites load plenty of scripts, styles, and images and perform various AJAX calls on load so it is not any problem to store for instance a 32-bit identifier in this way. With proper names of the subdomains it can look like load balancing at the first look.

This way of storing supercookies may look too complicated for practical use – it requires a wildcard certificate and more subdomains to work and there are much simpler and cheaper methods like ETag. However, the main power of this method is that it can potentially track the user even if she uses the anonymous browsing mode [61]. It is a well-known fact and authors of each browser had to decide whether more important user privacy or security is. Technically, there are two options – to use the HSTS cache in the anonymous mode and therefore risk that the user might be tracked by an HSTS supercookie, or not to use the HSTS cache in the anonymous mode and risk that there can be attacker using the SSL Strip method. There is no official descrip-

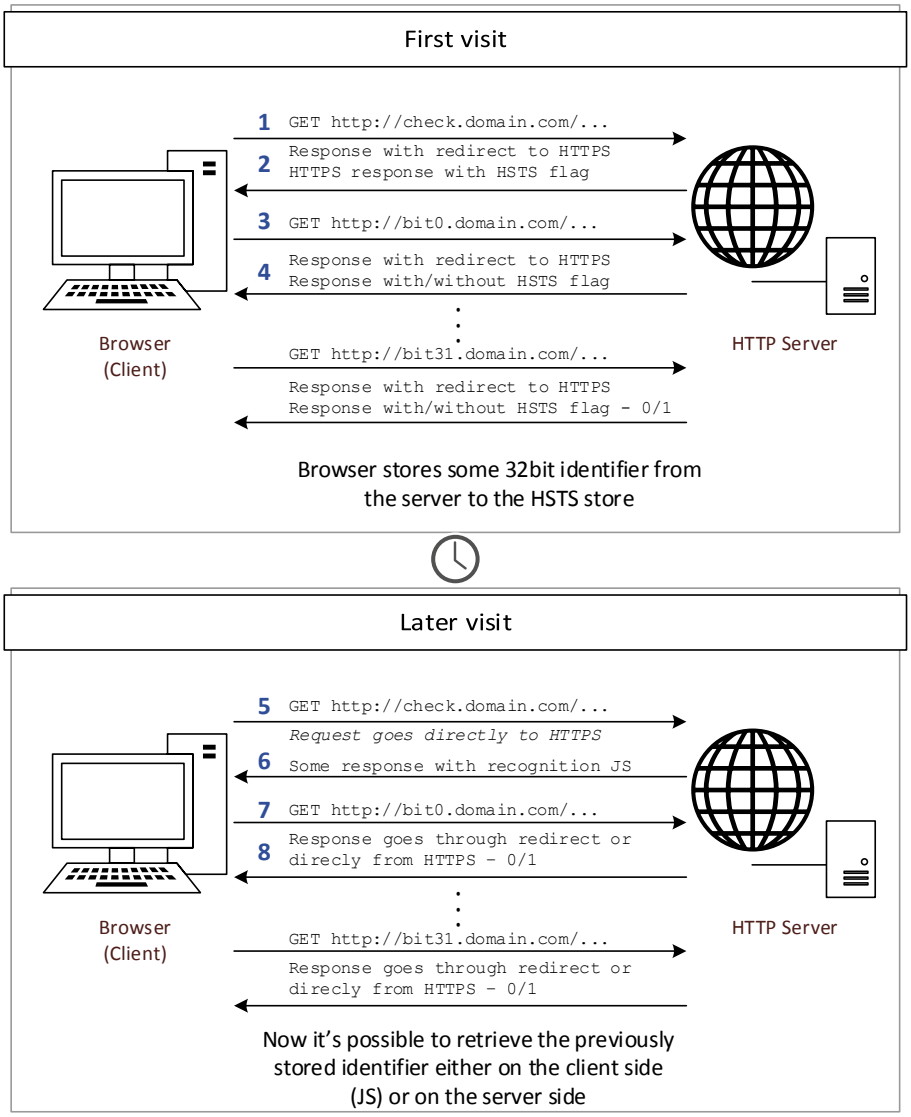

Figure 3.7: HSTS tracking mechanism

First, a website needs to check whether a user has been already marked by the HSTS tracking technique (1,5). It is done by sending a request to a special subdomain, serving only for that purpose. If the request is redirected by the browser to `https://...`, the user already has been marked (6), if not, then the user has not been marked yet (2). Eventually, requests are sent to many special subdomains (3,7), each request works with a single bit of information. These requests serve either to set a bit value by returning or not returning the `Strict-Transport-Security` header (4), or to read the bit value by checking whether the request was redirected by the browser or not (8).

tion of the current browsers behavior to HSTS in anonymous mode, it will be tested as a part of this thesis.

3.2.4 HTTP Public Key Pinning

The secure communication channel is crucial for many applications. When an application uses the Public Key Infrastructure (PKI) [62] to establish secure communication, it usually has to trust the Certificate Authorities (CA) [42] and the Domain Name System (DNS) [43] servers. These subjects can be compromised in the user's environment or they can theoretically be compromised at all, which can eventually break the application security. For that reason some applications pin their certificate or public key [63]. They use standard PKI process but after receiving the public key they perform an additional internal check of the returned certificate. For instance, according to the practical testing [64], the App Store application on iOS devices performs check against the pinned certificate when it sends the user credentials to a login endpoint, failing by rejecting the server's certificate when it does not match the pinned one.

The HTTP Public Key Pinning (HPKP) is defined in RFC 7469 [40]. It provides an option for the server to send to a client a hash of the Subject Public Key Info (SPKI), part of the X.509 certificate [65], which has to be present in a web host's certificate chain to consider it valid. The hash (or more hashes) is returned in a `Public-Key-Pins` header of an HTTP response. The hash is then stored on the client side and it is used to validate all following requests. Therefore the HPKP mechanism, same as HSTS, supposes that the first request is performed through a secure link. In specific cases the certificate hash can be a part of a browser installation. For instance, the Google Chrome web browser pins the certificate of Google applications like Gmail [66] in its installation.

The `Public-Key-Pins` header has to contain, along with the mentioned hash of the SPKI, the `max-age` parameter telling the client for how many seconds should the retrieved hash be stored in the browser. There is a recommendation in RFC that the browser should not store the hash for more than 60 days, even when the retrieved `max-age` is higher. The `Public-Key-Pins` header, as same as the `Strict-Transport-Security` header, can also contain the `includeSubDomains` property, telling the client whether the hash should be pinned for all subdomains or for the requested domain only.

3.2.4.1 HTTP Public Key Pinning Abuse

The HTTP Public Key Pinning can be abused for tracking in the similar manner to HSTS. In case of HSTS, on the first visit requests are made to many subdomains, each either returns or does not return an HSTS flag – a bit set to 0 or 1 for each subdomain. On the next visit the requests are made again

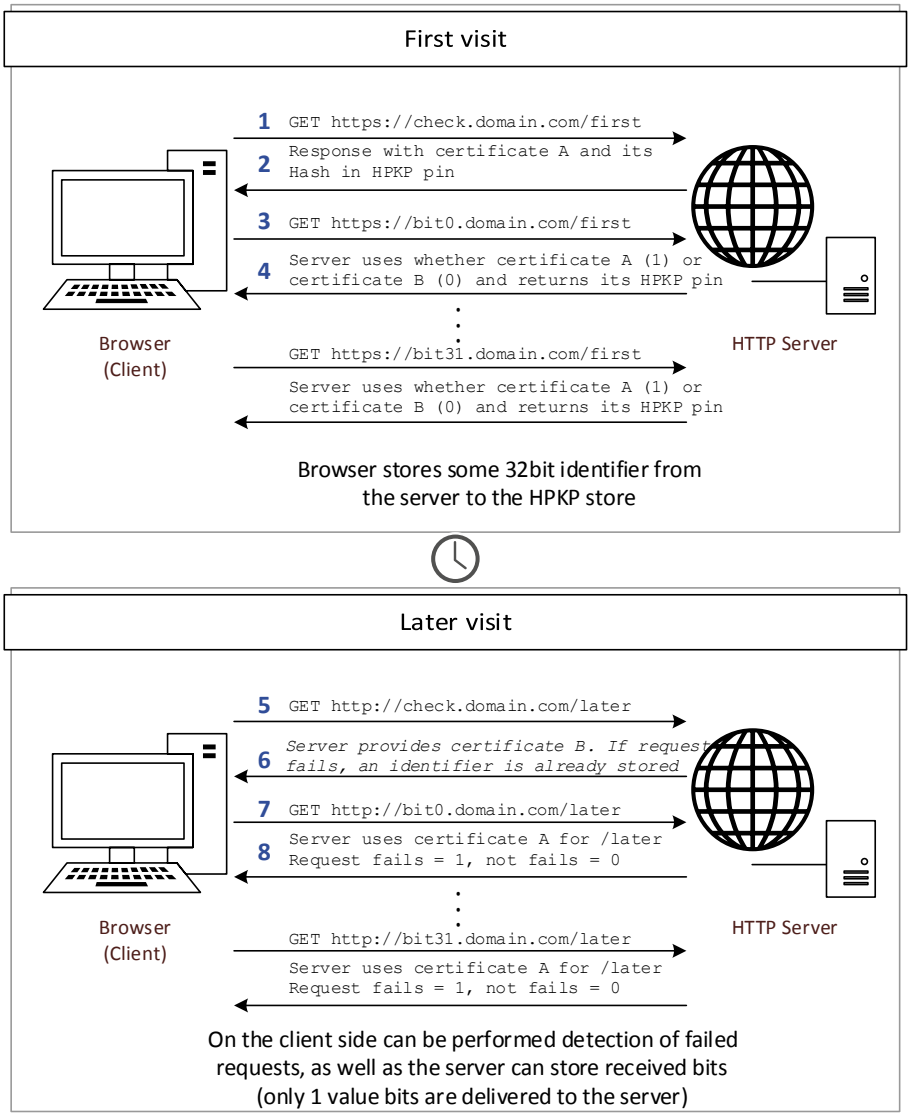

Figure 3.8: HPKP tracking mechanism

When a user visits a website the first time, in JavaScript is, based on a stored check domain certificate hash (server must have two wildcard certificates – A and B), decided whether an identifier is already stored or not (1,5). If there is no identifier stored yet (2), a new one is stored by requesting special URLs (3) which present one of two available certificates depending on the identifier bit value. The `Public-Key-Pins` header is returned along with each response, instructing the browser to remember the used certificate hash (4). Next time the user visits the website, initial checks fails on the certificate pin (6) and all subdomains are requested again using a URL which present itself always with the certificate A (7). If the request succeeds, the stored hash matches the certificate A and a bit value is considered as 1. If the request fails, the certificate B was pinned and bit value is 0 (8).

and browser redirection to HTTPS is detected in JavaScript, reading zeroes and ones back. A similar mechanism can work by providing two different valid certificates (Certificate A and B) for each subdomain (or two wildcard certificates) using the `Public-Key-Pins` header to pin the single subdomain certificate when the identifier is stored. On the next visit stored certificate hashes are checked by performing a request to a URL which always presents the same certificate (Certificate A) and depending on the pinned certificate hash (request fails or does not fail) is a bit value evaluated as 1 (Certificate A) or 0 (Certificate B). Fig. 3.8 shows the described principle in detail.

Tracking using HPKP supercookies is very similar to tracking using HSTS. It works on similar principle and provides similar results. However, it is more difficult to implement and deploy HPKP tracking, because two valid certificates signed by a trustworthy certification authority are needed for each subdomain. Another option is to have two valid wildcard certificates signed by a trustworthy certification authority and use subdomains covered by the certificates.

3.2.5 Adobe Local Shared Objects (Flash Cookies)

Although `HTML5` partially replaced Adobe Flash on websites, most desktop browsers still support this technology because of compatibility with old applications. However, Flash is not supported on mobile devices well – Apple declared in 2010 that its devices will not support Adobe Flash [67] and Adobe Flash for Android is no longer developed since 2011 [68].

Adobe Flash applications, as same as other applications, sometimes need to store their data somewhere. For that purpose ActionScript, a programming language for Adobe Flash applications, contains a mechanism called SharedObject [69], allowing applications to store user data either as a remote shared object, stored on the server and available to all users, or as a local shared object (`LSO`), a small file in the Adobe Flash plugin folder on the user's device.

Although local shared objects are sometimes called a "Flash cookies", they have the following basic differences:

1. More data can be stored in LSO than in a cookie. According to the Microsoft Developer Network (MSDN) [70], most browsers support cookies of up to 4 KiB. Local shared objects can contain up to 100 KiB of data by default.

2. Cookies are stored in browser storage, while LSOs are stored in the user data directory in the Adobe Flash plugin folder.

3. LSO does not expire.

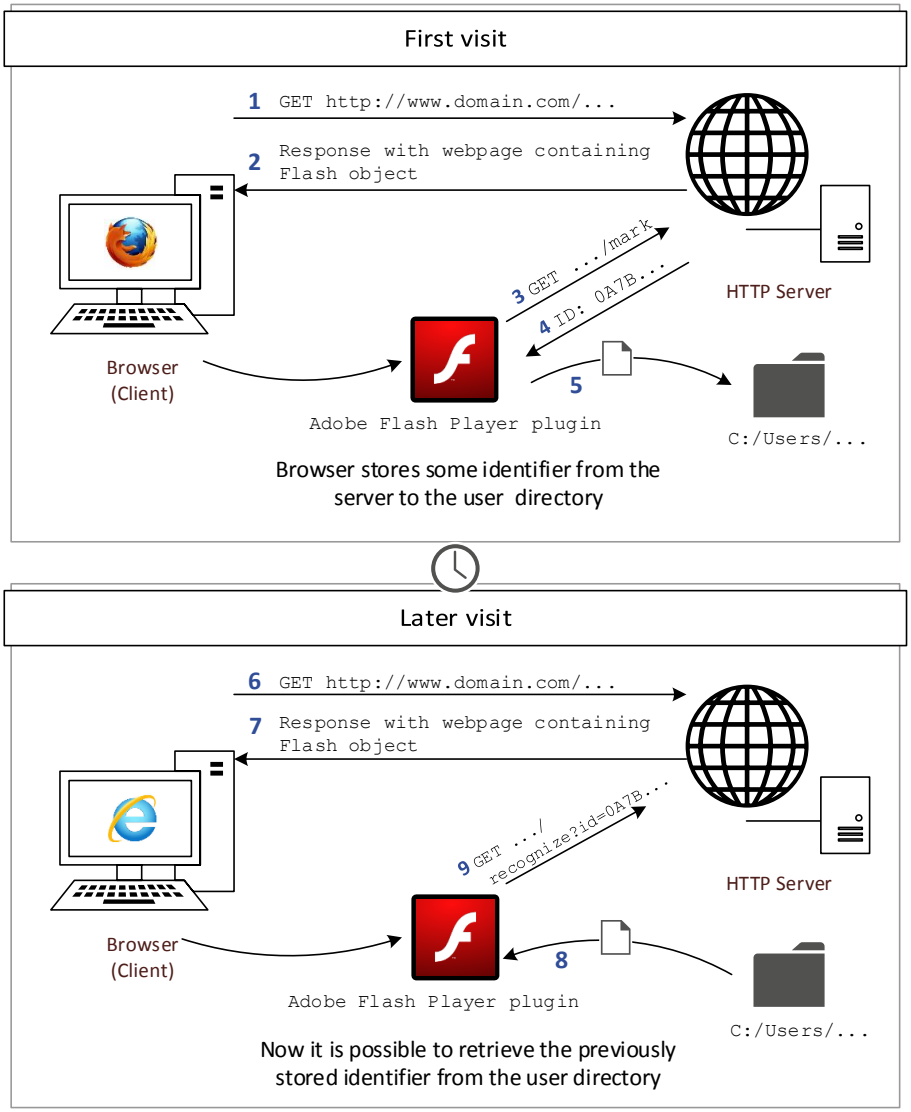

Figure 3.9: Adobe local shared objects tracking mechanism

When a user visits a website (1,6), an embedded Flash object is loaded (2,7) and its ActionScript code is run outside of the browser. On the first visit, it sends an HTTP request to the server (3) to retrieve an identifier (4), which is stored using LSO in the user directory (5). Next time the user visits the website, an identifier is loaded from the user directory (8) and sent to the server (9).

3.2.5.1 Adobe Local Shared Objects Abuse

The local shared object can be used for tracking users in the same manner as the HTTP cookie. It means that a web application stores an identifier generated on the server to the Adobe Flash plugin storage instead of or in addition to a standard HTTP cookie. Higher capacity of the storage does not improve its usability for tracking purposes but there is one great advantage over the standard HTTP cookie. Because the identifier is stored in the plugin folder rather than in browser storage, LSO can be used to track users across browsers. However, Adobe Flash Player applies the same-origin policy, which separates content from different domains into security sandboxes [71]. Therefore LSOs cannot be shared across domains. In 2009 came out that many websites use this method to track users [16].

3.2.6 JavaScript Browser Storage

JavaScript in the modern web browsers provides many functions to store data of the application in the browser. Most of these functions came with HTML5 to enable the applications to work offline. All of them can be used for tracking in the same manner as the regular cookie – the website just calls the JavaScript write function to store an identifier into the browser and later retrieves that identifier back using the JavaScript read function. The description of each method follows.

Window Name Every browser window or tab has its own `window.name` [72] property. It is usually empty by default, but it can be changed using JavaScript. The value is not persistent, but it can be used as a temporary identifier storage for tracking the user across more websites.

HTML5 WebStorage The concept of WebStorage [50] consists of two types of storage – LocalStorage and SessionStorage.

- **LocalStorage** HTML5 Local Storage [73] is a modern alternative to the cookies. It is more secure than cookies, because the values are not included in each request. It can also contain more data, at least 5 MiB. The local storage is per domain and it is persistent – data is visible from the same domain only and it is not deleted when the browser is closed.
- **SessionStorage** The session storage is similar to the local storage, the only difference is that data is deleted when the browser is closed. Therefore it is not as much useful for the tracking as the LocalStorage is.

HTML5 Web SQL Database Although the W3C Standard [74] is no longer in active maintenance, the modern browsers implement the web SQL

database using SQLite [75] database as storage. It can be used to store an identifier using SQL commands.

HTML5 Indexed Database API The HTML5 Indexed Database [76], shortly IndexedDB, is a newer W3C concept than Web SQL Database, which it replaces. The IndexedDB is not relational database, instead it only stores key-value data. It can store an identifier in an inserted key-value pair.

Browser Specific Storage Some browsers contain special JavaScript functions to store application data. An example of that function is userData [77] storage, working in the older versions of Internet Explorer. Some applications can potentially abuse this kind of functions to store an identifier but they have to perform a browser check before attempting to do so.

3.2.7 Other Supercookies Methods

There are also other methods to store an identifier, not mentioned above. They are not mentioned in a special chapter because they work only in specific cases or they have no perspective in the future. However, they should be mentioned for the sake of completeness.

The Silverlight [78] technology, developed by Microsoft, contains a feature called Isolated Storage [79], which enables applications to use a virtual file system on the client side. It is meant to store files, but inside of a file can be easily put an identifier. However, there has to be a Silverlight plugin installed in the user's browser to track the user by this method. Microsoft announced that the support for browser versions of Silverlight will end by 2021 [80], thus the tracking method will probably not work afterwards.

Another option is to abuse the browser history to store an identifier. On the first visit the browser retrieves some generated identifier from the server and stores it in the binary form by calling specific URLs, if the n^{th} bit is 1. The called URL can be, for instance `http://www.domain.com/n.html`. When the user returns to the website, links to all bit pages are rendered to page and in the JavaScript function is performed check whether user visited the URL or not – it works because the visited link usually has a different color than a not visited one. This method is called CSS History Hack [81]. However, countermeasures to this method are included [82] in the modern browsers nowadays, so the method only works in older browsers.

Java Applets can also store their data locally, using PersistenceService [83] interface. However, Java applets are usually disabled in modern browsers [84] by default and they have to be enabled manually. For that reason Java applets are not used much for the tracking.

As we can see, there are many known methods to store a supercookie in the user's device and we saw that their amount is growing in time as well. In fact,

there are probably much still unknown methods. However, there are projects mapping supercookies methods, like the Evercookie project [52]. We have already seen that some vulnerabilities were already fixed by browser vendors and there is a chance that their amount will grow as well.

3.3 Fingerprinting

In the previous sections we have presented various methods of storing identification data into the user's device. From the server's point of view we cannot know what will happen with the identifier stored in the user's device. It is difficult to cover all methods but still it is possible to delete all data after each request and to dispose of all stored identifiers [85]. It probably would make the browser less comfortable because of disabled content caching and maybe a little less secure because of disabled HSTS. Some applications dependent on cookies, Adobe local shared objects, or HTML5 Web Storage might not work properly. In this chapter will be presented, that it is possible to identify user even when she successfully disposes of all stored identifiers.

There exists also a converse approach – retrieving as much data about the user as it is possible and storing it on the server side. After the next visit the data is retrieved the same way again and compared to the stored one. When the match is found, the user is recognized and this information can be used to restore the cookie or just inconspicuously track the user's activity. If the fingerprinted information is combined with the user's behavior, it can be used to cross-device tracking as well [86]. Similar methods are also used by the malicious websites to detect the user's browser (browser fingerprinting) and decide how to perform an attack on the user's system [87].

According to research [31], in a user's device there are many attributes that can be abused for fingerprinting. The following chapter will describe how to calculate an amount of information in an attribute. Some of attributes are sent by the browser automatically in an HTTP request (passive fingerprinting), while some have to be read by running code on the client side (active fingerprinting). Both types will be described in the following sections.

3.3.1 Mathematical Treatment

The fingerprinting methods suppose that configuration of each user's device is much different from others making the fingerprint unique or almost unique. The amount of retrievable information for each fingerprint part can be described in bits as the uncertainty measure – the entropy. The entropy $H(X)$ of a random variable X with a probability mass function $p(x)$ is defined as [88]

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x), \tag{3.1}$$

where $p(x) \log_2 p(x)$ is defined as 0 if $p(x) = 0$.

Apparently, we need to know the probability of each fingerprint part value to count the entropy of the fingerprint part. We would have to store the values collected from a huge amount of users and analyze them to retrieve the statistical data needed to calculate the entropy.

We could use public statistics to calculate the entropy, but they are usually just derived from the fingerprinted fields. For instance, the `User-Agent` header field, described in section 3.3.2, contains a browser name and its version, the operation system and its version, and some other product versions or comments, while the available statistics contains just a browser name and its version. They can be used to calculate the minimal entropy of the `User-Agent` header field, but the result is far from the exact value.

Table 3.1: Browser version statistics from StatCounter (November 2015) [89]

Browser	**Usage [%]**
Chrome 46.0	27
Chrome for Android	15.37
Safari iPhone	6.67
IE 11.0	6.49
Android 0	5.21
⋮	⋮
Opera 34.0	0.01

The November 2015 browser version statistics [89] from StatCounter [90] can be used to count the minimal entropy of the `User-Agent` header as follows:

1. Statistics retrieved from StatCounter as a CSV file. In Tab. 3.1 is an example of retrieved data.

2. Substitution of the retrieved values (189 records) to expression 3.1:

$$H = -[0.27 \cdot \log_2 0.27 + 0.1537 \cdot \log_2 0.1573 + \cdots + 0.01 \cdot \log_2 0.01] \quad (3.2)$$

3. The result is 4.375 bits.

We calculated the entropy of the `User-Agent` HTTP header to be at least 4.375 bits. The entropy was calculated from the browser version information only, but the `User-Agent` header contains more information – at least the operation system and device identifiers. Unfortunately, we do not have the required statistics to calculate the entropy in a more precise way.

It is not possible to calculate the entropy of the fingerprint parts without collection of the detailed statistics. For that reason, we have to settle for the minimal entropy counted in the existing research, like the Panopticlick project [31]. However, web browser developers implemented several countermeasures [91] after the Panopticlick project publishing, therefore the entropy probably will not be exactly the same in current versions of the web browsers.

There is a question how to calculate the entropy of the whole fingerprint if we know the entropy of fingerprint's parts. The joint entropy of a pair of discrete random variables (X, Y) with a joint distribution $p(x, y)$ is defined as [88]

$$H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log_2 p(x,y). \tag{3.3}$$

We could use 3.3 multiple times for each fingerprint part or count the entropy of the whole fingerprint from entropies of its parts using the chain rule [88]

$$H(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} H(X_i | X_{i-1}, \ldots, X_1). \tag{3.4}$$

The conditional entropy $H(Y|X)$ is defined [88] as

$$H(Y|X) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log_2 p(x|y). \tag{3.5}$$

From the chain rule follows [88] that

$$H(X_1, X_2, \ldots, X_n) \leq \sum_{i=1}^{n} H(X_i) \tag{3.6}$$

with equality if and only if the X_i are independent.

In order to calculate the entropy of the whole fingerprint the particular statistics of fingerprint parts combinations are needed. It would not be needed if the parts were independent, but they obviously are. For instance, if the `User-Agent` header says that the request comes from an iOS or Android device, the probability that a Flash plugin is present decreases because Apple never supported Adobe Flash [67] and Adobe Flash for Android is no longer developed since 2011 [68].

Because the fingerprint parts are not independent, it is not possible to calculate the total entropy without detailed statistics. However, an existing research can be used to specify a lower bound of the entropy. If the implemented fingerprint contains all parts that are contained in a fingerprint used in an existing research, it can be observed that the entropy of the implemented fingerprint is greater or equal to the entropy of the fingerprint in the research. That is the only way to estimate the fingerprint entropy without detailed statistics.

3.3.2 Passive Fingerprinting

When a server uses passive fingerprinting methods, no code is run on the client side and information is retrieved from the HTTP request only. The HTTP request can contain some headers along with the request body. Most of possible HTTP headers occur in special cases only but there are few headers that are

contained in most requests. Although they usually do not contain enough information to make the user's device unique, they can be combined with other fingerprinting methods increasing total entropy. Below is listed information contained in the request which is the most useful for tracking:

`User-Agent` header The main source of fingerprinting information in the request is the `User-Agent` header, its current definition can be found in section 5.5.3. of RFC 7231 [92]. It should contain one or more product identifiers with an optional product version and a comment. It is usually possible to identify at least the browser and the operation system of a user thank to this header. According to the Panopticlick project, the entropy of the `User-Agent` header is 10 bits.

Accept headers There are four HTTP headers defined in the Content Negotiation section (5.3.) of RFC 7231 [92]: `Accept`, `Accept-Charset`, `Accept-Encoding`, and `Accept-Language`. Their purpose is to inform the server about the client's preferences, but they also divulge details about user's environment – the most obvious is the user operation system language in the `Accept-Language` header. According to the Panopticlick project, the entropy of the Accept headers is 6.09 bits.

IP address Although an IP address can change over time, especially in case of mobile devices, and there can be many devices behind a single public IP address, it can help as an additional fingerprint source in some cases.

The main advantage of passive fingerprinting is its inconspicuousness. Servers can analyze the request headers and clients do not have any chance to detect it. However, according to known research, the HTTP headers alone usually do not provide enough information to identify a user. The real HTTP headers entropy probably will not be much higher than 10 bits because the accept headers mostly depend on the browser and the operation system. For that reason the passive fingerprinting needs to be deployed together with the active fingerprinting to provide enough information to identify the user.

3.3.3 Active Fingerprinting

There is much information about the user that an opened website can retrieve from the web browser [32]. When code has to be run on the client side to collect required fingerprint parts, it is called an active fingerprinting. The code must be transferred from the server to the client, executed, and its results have to be delivered back to the server. Such attempt can be detected or blocked on the client side.

Active fingerprinting is mostly performed using JavaScript code. JavaScript is well supported across the web browsers and many websites cannot work properly with JavaScript disabled. There are still about one percent [93]

of users having JavaScript disabled – it can be either caused by a special device type, a blocking proxy, or a browser plugin like NoScript [94]. In that cases only passive fingerprinting together with information about disabled JavaScript can provide enough information to identify the user. However, according to the Panopticlick research, users with the NoScript plugin in their web browser are worse "fingerprintable" than others.

3.3.3.1 JavaScript Attributes

The simplest way to perform active fingerprinting is by just reading attributes accessible from JavaScript and send them back to the server. The most important sources of JavaScript attributes follow:

Navigator object The base source of the JavaScript attributes is the navigator object [95]. It contains information about the browser like its type, version, plugins, or the user agent string sent in HTTP requests. The standard [95] defines only 10 public attributes and 2 methods but the browsers usually add their own attributes and methods. For instance, the navigator object in Chrome exposes the total number of logical processors in its `hardwareConcurrency` [96] property. According to the Panopticlick project, the most important navigator object property for fingerprinting is the `plugins` property, providing 17.7 bits of entropy in the browsers where it is supported. It provides a list of installed plugins and MIME types they support. However, Internet Explorer did not expose the `plugins` property in the navigator object before Internet Explorer 11. Firefox started to limit it in 2014 [97] due to its privacy impact [91] and the ending support for NPAPI plugins [98, 99] reduces a browser plugins count and the entropy as well. If enumeration is not supported, presence of a plugin has to be tested by its name. The prepared list of plugin names is needed to get a list of plugins in user's web browser by checking each one's presence.

Screen object The attributes of the device display can provide another bits of information. They are accessible through the screen object [100]. For fingerprinting purposes it is useful especially the screen resolution, contained in the `height` and `with` attributes of the screen object. However, it can change when an external monitor is connected. The screen object can be useful mainly for tracking users of the mobile devices, where an external monitor is rare and screen resolutions vary much [101]. According to the Panopticlick project, the entropy of the device display parameters is 4.83 bits.

Date object Another bits of information can be obtained from the date object [102]. It provides the current date and time at the client's device as well as the time zone offset. The Panopticlick project stated that

the `timezoneOffset` property of the date object provides 3.04 bits of entropy.

Technologies presence JavaScript can be also used to check if other technologies are enabled or present. The already mentioned `plugins` property of the navigator object can provide information about installed plugins and its version. In addition, the navigator object contains the `cookieEnabled` [103] property and the `javaEnabled` [104] method that can be used to check whether cookies or Java are enabled. According to the Panopticlick project, the entropy of the `cookieEnabled` property is 0.353 bits.

In general, the JavaScript programming language provides many ways to get information about a user's device or system. The most significant ones are listed above, especially the `plugins` property of the navigator object deserve due consideration. Other attributes are not quite useful alone but combining them with the other ones and with passive fingerprinting can work well, creating a unique fingerprint of the user's environment. JavaScript attributes can be also used to correctly identify [105] the user's environment when the `User-Agent` HTTP header is modified, for instance when using Tor [106].

3.3.3.2 Font Fingerprinting

System fonts in the user's device can be also used to generate a fingerprint. A list of installed fonts is influenced by the operation system, installed software, and manually installed fonts. According to the Panopticlick project a list of fonts can provide 13.9 bits of entropy, not concerning the font order.

There are four ways to retrieve user's system fonts from a web application:

CSS + JavaScript Font Detection Because the CSS `font-family` [107] property allows developers to provide a list of font names, where the first one available in the system is used, it can be used together with JavaScript code to detect a font presence in the user's system. The detection mechanism creates a block element with a string consisting of significant font characters. First, its font-family property is set by JavaScript to a generic font-family like "sans-serif" or "serif". Width of the element is then checked by JavaScript. Next, the font-family is set to a checked font name and the generic font as the second one in the priority list. The width is measured again – if it is equal to the width measured before with a generic font, the checked font is probably not present in the system (the element style fell back to the generic font). The described method cannot be used to get an exact system fonts list, it can be used to check a presence of the specified font only. Therefore a prepared font list is required. There are several reference projects [33, 108] using this method to list the user's system fonts.

Adobe Flash The ActionScript language, the scripting language for Adobe Flash applications, contains the `Font` class with its `enumerateFonts` method [109]. It can be used to retrieve an installed font list in a Flash application function, either the result is passed to the JavaScript code [110] or the Flash application sends data directly to a server. The fonts' order can be also used as a fingerprint part. It is worth mentioning that the Adobe Flash plugin can be configured to disable the font enumeration using the `DisableDeviceFontEnumeration` configuration option [111]. The font enumeration is enabled by default, it has to be disabled manually.

Silverlight A list of installed fonts can be loaded using the `SystemTypefaces` property of the `System.Windows.Media.Fonts` class [112] when programming in the Silverlight language. It works similarly as in case of Adobe Flash. An ordered list is returned and can be either accessed through the JavaScript API for Silverlight [113] or sent directly to the server. As already stated, Silverlight support will end by 2021 [80].

Java Applet The Java language contains the `GraphicsEnvironment` class, providing the `getAvailableFontFamilyNames` [114] method, that can be used in a Java applet to retrieve an array of available system fonts. The returned structure is a plain Java array, therefore it provides an order of the retrieved fonts as well. The result can be passed to a JavaScript method using the `JSObject` class [115] or sent directly from the Java applet to the server.

Font fingerprinting is useful in case of the desktop operation systems where it is usual for installed applications to add a font to the system. The fingerprint can evolve in time as the user installs or uninstalls applications, thus the retrieved fingerprint should be compared with the stored one considering the possibility of this kind of change. Font fingerprinting is not much useful on mobile devices where the system fonts are usually not changed by installed applications.

3.3.3.3 Battery Fingerprinting

One of the features presented in HTML5 is the Battery Status API [116]. It is a JavaScript API allowing web applications to check for the device battery status. Its purpose is to enable web applications to switch to an energy saving mode when the battery is low. Apparently, it works on mobile devices with batteries only.

The Battery Status API provides not only a battery level, but also a charging time, discharging time, and an indicator whether the battery is charging right now. In addition, each of these attributes can be watched by registering to one of the listeners provided by the Battery Status API.

Recent research proved [117], that the revealed information can be also abused for tracking the user. It can be helpful mainly in a short time interval, it can reveal a switch to anonymous mode, cleared cookies, or eventually deleted supercookies. It can also link the visits to different sites in a short time interval.

The research also shows that the battery capacity could be calculated precisely from information retrieved using the Battery Status API. However, it was tested on the Firefox browser on Linux, that provided high precision values of attributes and Mozilla already deployed a fix providing less precise values.

The event listeners provide an option to measure how fast the capacity of battery is drained or recharged. The information can be used to calculate unique battery properties [118] and abuse them for tracking.

The number of mobile devices grows rapidly with the new technologies. There are not only smartphones and notebooks, but also tablets, convertible devices, Internet of Things (IoT) devices, and wearable electronic. All these devices use a battery. Therefore, battery fingerprinting has a large potential.

3.3.3.4 Canvas Fingerprinting

Using the preceding methods allows websites to retrieve information about the user's environment that should be processed and compared between the requests in the right way to work as a reliable user identifier. As opposite, canvas fingerprinting does not provide any specific information about the user's environment, rather it provides a way to generate an identifier dependent on more factors of the user's environment.

One of the features that came with HTML5 is the canvas element [55], enabling websites to draw graphics by JavaScript. It provides many methods to manipulate with graphics, including the 2D context `fillText` method to draw text and other methods to perform CSS-like text styling.

It is necessary to be able to read image data in order to create a fingerprint. The 2D context of the HTML5 canvas provides the `getImageData` [119] method returning an array of RGBA values of each pixel in the canvas content. The canvas object itself provides `toDataURL` and `toBlob` methods [120] that can be used to retrieve data. They both accept a parameter of a required image type, like `image/png`, the first one returns image data in the Base64 format, while the second one returns it as raw data in a Blob object.

HTML5 canvas enables developers to create large graphic applications in the browser. For instance, the canvas can be used to create advanced games or online photo editors. Because this kind of applications requires much performance to run, modern web browsers use graphics processing unit (GPU) acceleration [121] to draw a canvas object. This is the main reason why canvas fingerprinting works and why it provides different results across environments.

The research from the University of California [122] examined possibilities of canvas fingerprinting. They have used the `fillText` method of a 2D context to fill in a styled text and the WebGL JavaScript API [123] to draw a simple three-dimensional object to the canvas object. They performed tests with different GPUs, browsers, and operation systems and discovered that the method can provide up to 5.73 bits of entropy.

Canvas fingerprinting does not require any special plugins or permissions, it uses just the JavaScript language and HTML5 features. Therefore it can be used in most user's environments. It can be limited only by the old version of a browser, manually disabled hardware acceleration, or a graphic card not supporting WebGL.

The retrieved fingerprint has good stability because users usually do not change their graphic cards often. The fingerprint may change when the graphic card driver is updated, but it also is not quite usual. It can be influenced by the graphic card switching on systems with more than one graphic card, but user would have to do it manually, because the browser would probably use the same graphic card on each visit by default.

As already stated, canvas fingerprinting depends, among others, on the GPU and its driver. Because any of the previously mentioned methods do not depend on the GPU, canvas fingerprinting can be used to increase the entropy of the whole user's fingerprint significantly.

3.3.3.5 Browser History Fingerprinting

Web browsers usually store history of visited websites. Apparently, a chance that two users would have exactly the same browser history content is slight. If a website is able to read the user browser's history or at least its part, it can serve as a user's fingerprint.

Section 3.2.7 noted, that in older browser versions it was possible to check whether the user had visited a URL in the recent past by checking a link element's CSS style, abusing the style change of visited links. Research [124] proved that this method was deployed on some popular websites, leading web browser vendors to implement countermeasures in 2010 [82]. In 2010 it was also presented a proof of concept [125] of user's identification based on her membership in groups at the popular social networks, abusing this user's history leakage.

Web browsers fixed [82] their behavior so that the visited links style is indifferent from the not visited one when accessed from JavaScript DOM methods, while the appearance of the links still remains unchanged. While the basic method does not work in browser versions released after 2010, there were invented other methods abusing the visited link style in various manners [126], which are nowadays perceived a vulnerabilities having impact at the user's privacy. The countermeasures can be overcame by forcing the user to reveal to the website what she sees. The CSS style can be abused to show special

content if the checked URL was visited before. From the user's reaction can be derived whether she saw the content on the website or not. There is also a pixel stealing method proof of concept [127], revealing whether the page was visited by the user or not by comparing the rendering framerate of the visited link and the not visited one.

Abusing the CSS style of the visited link to reveal the web browser history is in a broadly discussed topic last years, causing web browser vendors to fix reported vulnerabilities. However, the already mentioned (section 3.2.3) HSTS mechanism can be also abused [128] to check whether a user visited a website or not. However, this mechanism works only for websites using HSTS but it can still provide enough information to create a fingerprint.

Exposing user's browsing history alone is a very sensitive problem, causing the browser history leaking methods to evolve fast. A leakage method is presented, countermeasures are implemented by browser vendors, and a new method is presented to overcome the countermeasures or abuse another vulnerability. Over and over again. Such race causes nowadays working method to potentially stop working in future browser versions.

3.3.4 Other Fingerprinting Methods

There are plenty of JavaScript APIs accessible to the websites and their number is significantly growing with the HTML5 features. Each of these methods can potentially provide information about the user's environment, exploitable to create a fingerprint or at least increase the entropy of a fingerprint based on the previously mentioned methods.

Generating a fingerprint based on the user's hardware is an efficient way to track the user, as it is in case of canvas fingerprinting (section 3.3.3.3) and battery fingerprinting (section 3.3.3.4). It was proven [118] that other hardware like the camera, speakers/microphone, motion sensors, or GPS can be used to generate a fingerprint as well. To access hardware, the website must ask the user to grant a special permission. For that reason it can be used for fingerprinting in special cases only.

Another interesting hardware parameter is the processor. The number of processor cores is obtainable by a simple timing test [129] using JavaScript, even when the browser does not expose the information directly in the navigator object. The processor can also be used for fingerprinting using a special privacy attack based on measuring the CPU skew exposed through the TCP stack [130]. The information is exposed by the operation system on the TCP layer, so it actually is not a web browser issue. However, it can be used to increase passive fingerprinting entropy, because it does not require any special code to be run on the client side. Similarly, the GPU skew can also be measured and used as an entropy source [118]. In this case the measurement is done using JavaScript and can be used as part of the active fingerprinting.

We have seen that plenty of technologies having nothing to do with user's identity, actually expose information about the user's environment to the website. It can be expected that the number of such methods will grow with new browser features. However, there are projects warning against this privacy threat [31, 32, 131] and we have seen as well that web browser vendors react to published information and fix web browsers accordingly [91].

3.4 Summary

The chapter presented many methods capable of marking or remembering the user and recognizing her in the future. If a company really wants to track its website visitors, it has many options to choose from. The examined methods were divided to three sections: cookies, supercookies, and fingerprinting.

Cookies were discussed in section 3.1. They are the basic and well known concept of storing website state in clients' devices. Because they are used for tracking for a long time, they are well known and can be easily blocked. For that reason more advanced ways to track the user were invented – supercookies and fingerprinting.

Supercookies were described in section 3.2. They are various methods to store a generated identifier on the client side and retrieve the identifier back in the later requests. Though there are many places to store the identifier, it is difficult but possible to delete the identifier from all these places and therefore avoid the tracking. It was the reason to develop the tracking method that works even when the user deletes all data in her profile – fingerprinting.

Fingerprinting abuses a fact that when a user opens a website in her web browser, the web browser actually exposes much information that if combined properly can work as a natural identifier of the user. Various methods of collecting the information were described in the Section 3.3.

The next chapter will describe an implementation of the selected methods for the showcase included in the thesis. The implemented methods will be tested in the most used web browsers and test results will be included in the chapter.

Chapter 4

Implementation and Testing

Selected methods analyzed in the previous chapter will be demonstrated in a showcase. This chapter will describe implementation details and discuss the results obtained from the testing.

The server side of the showcase will be implemented in the Java programming language, using the Spring Framework [132]. It enables developers to simply create servlets, manipulate with HTTP headers, and perform database operations. The jQuery JavaScript framework [133] will be used on the client side to manipulate with HTTP headers and to perform AJAX requests. The showcase is designed to have minimum prerequisites, therefore user identifiers and fingerprints are stored in an embedded H2 database [134]. The application logic is targeted at the Wildfly server [135], enabling anybody to run the showcase and setup the environment using a single Maven [136] command: `mvn wildfly:run`, not requiring any setup, except for having Maven and Java installed. However, for testing the HSTS supercookie method (requiring HTTPS and many subdomains) the localhost subdomains have to be added to the hosts file and the server root certificate has to be imported into the web browser. A manual can be found in appendix B or on the enclosed CD.

The new user identity will be generated on opening the welcome page of the showcase. The identity will contain an easy to remember nickname, visible to the user, and its generated alphanumeric identifier used when applying implemented methods, hidden in the page. There will be two buttons rendered for each implemented method. The first one will run the marking or remembering process of a new visitor, applying a selected method to persist the current identity, represented by the generated identifier hidden in page assigned to the visible nickname. The second one will run the recognition process of a returning visitor, showing the stored nickname, if recognized. The real application would probably first run the Recognize phase using all methods and then it would eventually store a new identifier. However, in the showcase the two phases will be divided for better illustration.

The current version of the three most used desktop browsers in November 2015 [137] will be used for methods testing. A list of used browser versions follows:

1. Google Chrome 47 (47.0.2526.106)
2. Mozilla Firefox 43 (43.0.3)
3. Microsoft Internet Explorer 11 (11.0.9600.18124) – shortly MSIE

The anonymous web browser mode will be tested as well. It is expected that storing and reading identifiers in the anonymous mode should work the same way as in the normal mode, but the two modes should be separated strictly. If the value used in the normal mode is visible from the anonymous mode or vice versa, the method is considered as capable of circumventing the anonymous mode. The browsers will be running on Windows 7, currently (November 2015) the most used operation system, used by 45.2 % of website visitors [138]. Flash dependent methods will be tested on Adobe Flash Player 20.0.0.267 (Google Chrome and Mozilla Firefox) and 20.0.0.270 (Internet Explorer).

The chapter is organized similarly as the analysis chapter. First, the basic HTTP cookies implementation will be described, followed by supercookies methods implementation description, and enclosed by the fingerprinting mechanisms implementation description. Each method alone will be tested in the browsers and the results will be compared in the final summary.

4.1 Cookies

Because the cookie is a well-known and longtime used technology, most programming languages and frameworks provide functions to work with it. The `Set-Cookie` header value does not have to be composed manually, special classes and methods generate it automatically in the right format. The Java language provides `javax.servlet.http.Cookie` class to manipulate with cookies. List. 4.1 shows how can such a class be used to add the `Set-Cookie` header to an HTTP response.

```
1 String cookieId = userIdentityService.randomIdentifier();
2 Cookie cookie = new Cookie(Constants.COOKIE_NAME, cookieId);
3 cookie.setPath("/");
4 cookie.setMaxAge(Integer.MAX_VALUE);
5 response.addCookie(cookie);
```

Listing 4.1: Setting a cookie in Java

The preceding code instructs the server to send response with the `Set-Cookie` header like the one below.

`Set-Cookie:` tracking_showcase_cookie=deljrkd25j3r6cgao23ttk70j0; path=/;
Max-Age=2147483647; Expires=Thu, 10-Jan-2084 21:30:46 GMT

In the later request is the `Cookie` header in an incoming HTTP request automatically parsed by the servlet and accessible as a Cookie object again. Similar methods are available in other programming languages as well.

4.1.1 Cookies Testing

Saving a user's identifier in the cookie is easy to implement as well as to detect or prevent. As expected, the cookie works well across web browsers and it is dealt with correctly in the anonymous mode.

Table 4.1: Cookie support testing

Browser	Works in normal mode	Can circumvent anonymous mode
Chrome	✓	
Firefox	✓	
MSIE	✓	

4.2 Supercookies

Using the supercookie to track the user consists of storing a generated identifier somewhere in the user's device. Thus all demonstrated methods will have two phases (buttons). In the first one (Mark) the supercookie will be stored in the device, while in the second one (Recognize) a reading attempt of a once stored supercookie will be made. All methods presented in the thesis will be implemented, except for the HPKP supercookie. From two similar methods abusing HTTPS security mechanism in similar manner (HSTS and HPKP supercookie) was selected, after agreement with the supervisor, the HSTS supercookie for implementation, because the HPKP supercookie method is not convenient for localhost deployment.

4.2.1 Cached Resources

Storing a supercookie value in the cached content does not require any special technology. An identifier is stored in the content of the returned resource and the browser is instructed to keep the resource in the cache using the `Cache-Content` header.

Section 3.2.1 noted that the identifier can be stored in any kind of the cached content. The only difference between these kinds is the encoding. In some kinds of encoding the identifier can be better hidden, making it more

difficult to notice the identifier in a resource. However, the resource type does not have any impact on its tracking ability. Therefore it is sufficient to use a simple static variable in a JavaScript resource to test this method.

In the Mark phase an AJAX request is performed, loading the JavaScript resource from the specified URL. The server has to be informed about the user's identity, but this cannot be done by a request parameter because the parameter could cause the content not to be cached. Hence, the identity of the user is placed in a non-standard HTTP header `userId`. This approach allows the server to retrieve the nickname of the user that clicked on the button, while the caching function of the browser is not influenced. In the request there is also placed a `tmp` header, which contains a random value which is just copied to the response on the server side. This mechanism serves to detect whether the content was returned from the server (`tmp` headers match), or from the browser cache (`tmp` headers do not match). Once a resource is cached, it cannot be deleted from the browser cache using JavaScript, it has to be deleted manually by the user. List. 4.2 shows the described principle.

```
function cachedFirst() {
  var tmp = Math.random().toString(36).substr(2);
  $.ajax({
    url : "first/cached",
    beforeSend: function(jqXHR){
      jqXHR.setRequestHeader('userId', userId);
      jqXHR.setRequestHeader("tmp", tmp);
    }
  }).done(function(data, textStatus, jqXHR) {
    if(jqXHR.getResponseHeader('tmp') != tmp)
      showResult("Please clear your browser cache before running the test.");
    else
      showResult("The identifier of <strong>"+userNicname+"</strong> was succesfully stored in the cached content.");
  });
}
```

Listing 4.2: JavaScript part of storing a supercookie in the cached content

In the Recognize phase the same URL is requested again from the browser and the global variable contained in the JavaScript file is read. Apparently, the `userId` is not sent this time, therefore if the resource is not cached, the global variable does not exist and the recognition failure is reported. If the global variable exists, its value is sent to the server to retrieve the nickname associated with the value.

4.2.1.1 Cached Resources Testing

The cache is a crucial part of each web browser making website browsing comfortable and fast. Hence it is no surprise that the method works well in all web browsers. The anonymous mode's purpose is, among others, to hide

a user's browsing from other users of the same device. Besides clearing the history, the anonymous mode clears the cache content when the web browsing session ends. It was proven by testing that the anonymous mode in all tested browsers does not use resources cached in the normal mode and vice versa.

Table 4.2: Cached resources supercookies testing

Browser	Works in normal mode	Can circumvent anonymous mode
Chrome	✓	
Firefox	✓	
MSIE	✓	

4.2.2 HTTP ETag

Storing a supercookie in the entity tag (ETag) abuses the browser cache, as well as storing the supercookie in the cached content. Hence the implementation is analogous to the previous case.

Mark phase consists of performing an HTTP request to a specific URL, placing the user's identity into the `userId` header. An ETag identifier is generated and saved on the server side and returned in the `ETag` header. The contents of the returned file is irrelevant. The returned ETag identifier is then stored in the browser cache and nothing more is needed for the method to work.

When the Recognize button is clicked, a new request is performed to the same URL as when marking the user. The browser has the content from the URL stored in the cache along with the ETag identifier. Before the browser can return the cached content, it has to check on the server whether there is a newer version or not. When the version check is performed, the stored ETag identifier is sent to the server in the `If-None-Match` header. On the server, an ETag database is searched for the identifier and the user's nickname is printed in the response, and a new ETag identifier is generated, stored, and returned to the client.

4.2.2.1 HTTP ETag Testing

As already mentioned, the ETag supercookie is stored in the browser cache, same as the cached content supercookie. Apparently, testing ends with the same result as in case of the cached content supercookie.

Table 4.3: HTTP ETag supercookies testing

Browser	Works in normal mode	Can circumvent anonymous mode
Chrome	✓	
Firefox	✓	
MSIE	✓	

4.2.3 HTTP Strict Transport Security

Abusing the HTTP Strict Transport Security (HSTS) consists of forcing the browser to remember the HSTS flag for a subdomain, storing a single bit of an identifier. Later, the identifier is read back bit by bit by requesting all subdomains and detecting whether the request was performed through HTTP first or redirected by the browser directly to HTTPS.

Because the HSTS supercookie abuses the HTTPS mechanism in the web browser, the test server must present itself with a certificate. In the prepared showcase the Wildfly server is automatically configured by a Maven script to support HTTPS using a wildcard certificate for the `*.tracking.test` domain. However, to allow the web browser to communicate with the server, the public key of the certification authority from the showcase has to be imported into the browser and into the operation system store as a trusted root certification authority. In addition, the HSTS supercookie needs to use a plenty of subdomains to work properly. A simulation on the localhost can be realized by adding subdomain records into the hosts file, assigning them all to the `127.0.0.1` address. The required steps are described in appendix B.

The mechanism of storing the HSTS supercookie in the web browser is implemented in the way described in the section 3.2.3 and the communication between the web browser and the server follows the schema at Fig. 3.7, keeping the same subdomain names and storing a 32-bit identifier using 32 different subdomains.

When the Mark button is clicked, the following actions are taken. Note that all requests contain the `userId` parameter to identify the current user who should be remembered. Her nickname is printed on the page.

1. The `http://check.tracking.test` URL is called. When the browser redirects the request directly to the `https://` address, there exists an identifier already stored in the browser. In that case the HSTS flags have to be deleted before marking can be done. The deletion is made by sending the `delete` parameter to the server causing it to send `max-age=0` in the HSTS flag. The browser deletes its HSTS records and new marking can be done. When there is no identifier stored, the new one is generated

and stored on the server side, waiting to be read bit by bit using the subdomains requests.

2. The 32 asynchronous HTTP requests are generated, each on the corresponding subdomain. The server uses the retrieved `userId` parameter to load the previously generated identifier from the database. From the request URL it is retrieved a required bit number and depending on its value in the identifier is returned either redirection to HTTPS and an HSTS flag, or just an empty response.

On the Recognize phase, the same subdomains are used, detecting the internal browser redirection, reading the HSTS values stored in the browser back, consisting of the following steps:

1. The `http://check.tracking.test` URL is called again. When the browser redirection does not occur, a warning is shown, informing the user that no identity is stored and no other steps follow. If the redirection occurs, the second step follows.

2. There are 32 asynchronous HTTP requests generated, each on the corresponding subdomain. The server returns the zero value in each response to the HTTP request, or a corresponding power of two when it is requested through HTTPS. All results are summed up in JavaScript. The program has to wait until all responses are received before proceeding to the third step.

3. The calculated sum (the identifier) is sent back to the server. The identifier is then searched for in the database. If a match is found, the nickname of the recognized user is returned in a response. If not, the user is informed that recognition could not be done.

4.2.3.1 HTTP Strict Transport Security Testing

As already stated, the web browsers have to trust the used certificate authority in order to access HTTPS URLs on the application server. During testing it came out that the HSTS supercookie can also be blocked by browser protection against the Cross-site Scripting (XSS) [139]. When the user opens a website from the `http://tracking.test/showcase/` URL and the site performs an AJAX request to the `http://bit0.tracking.test/...` URL, it actually looks like an XSS attempt. It can be solved by adding the Cross-Origin Resource Sharing (CORS) [140] headers or using the JSONP technique [141]. For better compatibility, both methods are used in the showcase.

The HSTS supercookie works well in the normal mode of all tested browsers. However, testing revealed that each browser's behavior to the HSTS supercookies in the anonymous mode is different:

Table 4.4: HSTS supercookies testing

Browser	Works in normal mode	Can circumvent anonymous mode
Chrome	✓	✓
Firefox	✓	
MSIE	✓	

Google Chrome It came out that about 67 % [137] users can be tracked using the HSTS supercookie even when they use the anonymous mode. Testing proved that when the browser receives the HSTS flag in the normal mode, the flag is visible in the anonymous mode as well. It does not work in the opposite way. However, when the new HSTS flag is received in the anonymous mode, it is stored and visible from the anonymous mode only. It looks like there are two divided storages. In the normal mode, only the HSTS flags stored in the normal mode are used. In the anonymous mode, the HSTS flags stored in the anonymous mode are used in the first place, but when there are not any HSTS flag for the domain, the HSTS flag retrieved in the normal mode are used. However, the anonymous mode's HSTS flags are deleted when the browser window is closed, so the HSTS flag from the normal mode is visible on the next start of the anonymous mode. The Clear browsing data dialog deletes HSTS flags on submit, no matter what the user selects. It was tested, that even when deleting passwords from the past hour is selected, HSTS flags are deleted as well.

Mozilla Firefox browser treats HSTS flags the same way as, for instance, cached resources. The identifier assigned in the normal mode is not visible from the anonymous mode and vice versa. The identifier assigned in the anonymous mode is deleted when the browser window is closed. The HSTS supercookie is deleted when the Site Preferences option is selected in the Clear All History dialog.

Internet Explorer supports HSTS flags only in the normal mode. The anonymous mode does not use any stored HSTS flags and ignores the received ones. Therefore in the Internet Explorer's anonymous mode it is not even possible to store the HSTS supercookie temporarily, having the browser windows opened. In the showcase this fact becomes evident – when the Mark button is clicked, HSTS flags are sent to the browser. Even if the Recognize button is clicked immediately, the user is not recognized. The browser in the anonymous mode easily ignores all received HSTS headers. HSTS flags are deleted from the web browser when the Temporary Internet files and website files option is checked in the Delete Browsing History dialog.

4.2.4 Adobe Local Shared Objects (Flash Cookies)

The ActionScript language, a scripting language for Adobe Flash applications, provides an API to store user data in device – local shared objects (LSO). For the LSO supercookie method, Mark and Recognize buttons are replaced by an Adobe Flash content looking similarly. When the Mark button is clicked, a click action is fired and an assigned ActionScript function is executed. The server can be called directly from ActionScript code (it suits the same origin policy) but for better demonstration is the server requested from a JavaScript method which is called from ActionScript. The server returns a generated LSO identifier which is then stored by ActionScript. List. 4.3 shows the ActionScript part of the Mark phase.

```
function fl_ClickToStoreLso(event:MouseEvent):void
{
  var lso:String = ExternalInterface.call("lsoFirst");
  var so:SharedObject = SharedObject.getLocal("trackingTest");
  so.data.supercookie = lso;
  so.flush();
  ExternalInterface.call("lsoStored");
}
```

Listing 4.3: ActionScript storing an LSO supercookie

The ActionScript handles a click on the Recognize button as well. A value of LSO supercookie is read from LocalStorage and passed to a JavaScript function, which sends the value to the server and shows the returned nickname or a message telling that a user could not be recognized.

4.2.4.1 Adobe Local Shared Objects Testing

The described method uses Adobe Flash technology, thus its behavior depends on the Adobe Flash plugin rather than on the web browser. For the Firefox and Internet Explorer browsers the Adobe Flash Player has to be downloaded and installed in order to run Adobe Flash SWF files, while the Chrome browser contains the plugin in its basic installation.

Table 4.5: LSO supercookies testing

Browser	Works in normal mode	Can circumvent anonymous mode
Chrome	✓	
Firefox	✓	
MSIE	✓	

Testing revealed that Adobe LSO behavior in the anonymous mode does not differ from behavior of regular HTTP cookies. All three browsers delete Flash cookies when a cookie option is selected in a clearing browsing data

dialog. Because LSO values are actually not stored in the browser, rather they are stored in the plugin folder, they can be abused to track the user across web browsers, making the method remarkable for tracking purposes.

Table 4.6: LSO supercookies cross-browser testing

LSO	Obtainable from Chrome	Obtainable from Firefox	Obtainable from MSIE
Stored in Chrome	✓		
Stored in Firefox		✓	✓
Stored in MSIE		✓	✓

Table 4.11 shows that an LSO value stored in Firefox can be obtained from a Flash plugin running in Internet Explorer and vice versa. This is possible because both browsers use the same external plugin storing data in the same folder. The same origin policy is applied on the SWF file URL only, but the browser is not considered. Flash cookies are not shared with the Chrome browser because Adobe Flash objects loaded in Chrome actually run in a different Adobe Flash Player installation and thus they store their LocalStorage files in a different location than Adobe Flash objects loaded in Internet Explorer or Firefox. The same behavior can be expected in all cases where more browsers use the same Adobe Flash Player installation. However, the situation will probably change in the future because of Mozilla's plans to disable NPAPI plugins [98] and the Shumway Project [142], which should replace Adobe Flash in Firefox.

4.2.5 JavaScript Browser Storage

Several methods for storing a supercookie using JavaScript storage were mentioned in the analysis. Since all mentioned methods work on a similar principle, the WebStorage supercookie was selected for demonstration because it is a well-supported technology and it is proposed as a cookie replacement in the standard [50]. WebStorage consists of two types of storage – SessionStorage and LocalStorage. Only LocalStorage is suitable for storing a supercookie because content of SessionStorage is deleted when the web browsing session ends.

In the Mark phase an HTTP request to the server is performed, sending the current user identifier and retrieving the generated LocalStorage supercookie value. The retrieved value is then stored in the browser using LocalStorage. The Recognize phase consists of reading the LocalStorage supercookie value

and sending it to the server. On the server side the supercookie value is searched for in the database and if a match is found, a nickname is returned and displayed to the user. The user is informed as well if a match could not be found. List. 4.4 shows how the HTML5 LocalStorage can be easily used to store an identifier instead of a cookie.

```
function jsstorageFirst(){
  if(typeof(Storage) !== "undefined") {
    $.ajax({
      url : "first/jsstorage?userId="+userId
    }).done(function( data, textStatus, jqXHR ) {
      localStorage.supercookie = data;
      showResult("An identifier of <strong>"+userNicname+"</strong> was succesfully stored in a LocalStorage supercookie.");
      });
  } else {
    showResult("HTML5 LocalStorage is not supported by your web browser");
  }
}
```

Listing 4.4: Storing a LocalStorage supercookie in the JavaScript

4.2.5.1 JavaScript Browser Storage Testing

Since LocalStorage is meant to be a modern replacement of the cookie, it is correctly dealt with in the anonymous browser mode, the same way as with the cookie. All tested browsers delete LocalStorage content when cookies are deleted.

Table 4.7: LocalStorage supercookies testing

Browser	Works in normal mode	Can circumvent anonymous mode
Chrome	✓	
Firefox	✓	
MSIE	✓	

4.3 Fingerprinting

Fingerprinting methods consist of two phases, same as the previously mentioned methods (cookies and supercookies). Yet, the first phase is called Remember instead of Mark, because in this phase the user is actually not marked, only her environment properties are read and stored in the database. The second phase remains named Recognize – it reads the environment properties the same way as the first one and the results are compared on

the server. The comparison on the server side should be performed appropriately to correctly identify a returning visitor and to avoid any misleading results. It is a straightforward decision when the whole fingerprint exactly matches. However, as mentioned in the analysis, the fingerprint may evolve in time. When the user returns to the website, a little change in her fingerprint may occur, for instance the browser version number can be increased after a browser update.

Detailed statistics would be needed to construct a precise fingerprint evolution algorithm. Without statistics, we have to settle with a simple algorithm inspired by the Panopticlick project [31]. The collected fingerprint is stored as a serialized JSON [143] object in the database. In the Recognize phase a new fingerprint is collected the same way and transformed to the same object type as well. All stored objects are then converted to a form containing attribute values only, compared using the Jaro-Winkler distance metric [144], and their similarity is expressed as a percentage. Then the fingerprint with the highest percentage metric is used to recognize the user. If there are more stored identifiers with the same similarity percentage to the retrieved fingerprint, more identifiers are returned. If there is no fingerprint with more than 90 % similarity, the user is informed that there is no suitable fingerprint in the database. The 90 % is a guess that should cover new browser versions, a reconfigured DNT flag, a removed font, or similar changes. In a real application the guess should be correctly derived from collected statistics. Note that the testing implementation in the showcase is not much efficient because all data is retrieved from the database and then compared in Java. The main reason is that the data is stored in a simple embedded database which does not provide any advanced functions. Real applications would probably use advanced database functions like `utl_match` [145] in the Oracle database.

Fingerprinting methods described in the analysis chapter will be implemented in the showcase. All methods will use the comparison mechanism described above, except for the canvas fingerprinting method where the fingerprint evolution is not expected. After agreement with the supervisor, the battery fingerprinting will not be implemented and tested because it is applicable in special cases only (in a short interval and on mobile devices) and the implementation cannot provide any useful results.

The history fingerprinting method needs to have a list of websites prepared, supposing that the user has visited some of them recently. The list contents cannot be prepared generally for all websites, supposing that we do not want our website to perform thousands of requests. The list contents should rather depend on the website where the method is deployed. For instance, the list for a Czech IT magazine should contain other Czech IT magazines, few popular foreign IT magazines, and popular Czech websites. It can also contain the most popular websites like search engines or social networks, but they alone cannot provide enough entropy to recognize a single user. It is evident that the same list will not work for a different website type, e.g. an Italian

fashion company website. This fact, along with a questionable testability in our environment, and a fast evolution of the history leaking methods, was the reason to omit, after agreement with the supervisor, the history fingerprinting method from the showcase.

4.3.1 Passive Fingerprinting (HTTP Request Headers)

Passive fingerprinting needs to be implemented on the server side only. It easily stores selected request headers into a JSON object. The implementation uses the following headers: `Accept`, `Accept-Charset`, `Accept-Encoding`, `Accept-Language`, `User-Agent`, and `DNT`. An HTTP request with a `userId` parameter is performed from JavaScript in the Remember phase, causing the server to store the mentioned HTTP header values and assign them to the identity received in the `userId` parameter. An empty HTTP request is made on the Recognize phase to inform the server about the standard header values sent by the web browser. The retrieved values are then searched for in a database using the similarity testing mechanism described above. If a user is found in the database, her nickname is returned in the response. If not, a warning about no matching fingerprint is returned instead.

4.3.1.1 Passive Fingerprinting Testing

Passive fingerprinting is performed on the server side only, hence it is supported by all browsers. Testing proved that all tested browsers send in the anonymous mode exactly the same tested header values as in the normal mode. The passive fingerprinting can be then used to identify the user even when she uses the anonymous mode. It should be noted that passive fingerprinting of HTTP headers probably cannot provide more than 10 bits of entropy [31], thus the method alone cannot be used to identify the user. However, it can be useful in combination with other fingerprinting methods.

Table 4.8: Passive fingerprinting testing

Browser	Works in normal mode	Can circumvent anonymous mode
Chrome	✓	✓
Firefox	✓	✓
MSIE	✓	✓

4.3.2 JavaScript Attributes Fingerprinting

JavaScript provides many attributes useful for tracking the user. The following attributes were selected for the showcase: navigator object properties (`plugins`, `userAgent`, `language`, `appCodeName`, `appName`, `appVersion`,

and `product` properties), resolution (retrieved from the screen object), and timezone (retrieved from the date object). In the Remember phase these attributes are composed in JavaScript to an object, serialized into a JSON string, and sent as a GET parameter to the server. On the server side the parameter is deserialized back to an object, its format is verified, and the object is stored in the database.

The fingerprinted attributes are composed in JavaScript to an object again in the Recognize phase. The object is then sent to the server in the same way as in the Remember phase. The server uses the previously described fingerprint similarity mechanism to find the best matching user identity.

4.3.2.1 JavaScript Attributes Fingerprinting Testing

Fingerprinted JavaScript attributes were selected either from the standardized ones, or from the widely supported ones. Therefore the method works well in all tested browsers. Testing has proven that all tested browsers do not change any of the selected attributes when the anonymous mode is enabled. A fingerprint retrieved in the normal mode exactly matches a fingerprint retrieved in the anonymous mode in all three tested browsers. If JavaScript attributes fingerprinting is combined with the passive fingerprinting, it can likely be used to circumvent the anonymous mode.

Table 4.9: JavaScript attributes fingerprinting testing

Browser	Works in normal mode	Can circumvent anonymous mode
Chrome	✓	✓
Firefox	✓	✓
MSIE	✓	✓

From the data we have available it was not possible to correctly estimate an entropy of the method. The fingerprint contains the `plugins` property, which provides at least 15.4 bits of entropy according to the Panopticlick project. However, the browser cut plugins listed in the property in the last years [97], reducing the entropy. It is not possible to guess how much the entropy decreased without detailed statistics. It can only be stated that JavaScript attributes fingerprinting method provides at least 10 bits of entropy, since it contains the `userAgent` property.

4.3.3 Font Fingerprinting

There were four font fingerprinting methods described in the analysis. Three of them provide an exact result while the fourth one uses a single font presence check, requiring a prepared font list. A font reading using Adobe Flash was chosen for the showcase, because it provides exact results and the showcase

already required the Adobe Flash plugin's presence to store a LSO super-cookie. The Adobe Flash technology is also more used on websites than Java and Silverlight are [146].

The ActionScript programming language contains the Font class providing the `enumerateFonts` method [109] to return available fonts. List. 4.5 shows the `fl_GetFonts` function, retrieving all fonts, excluding Flash embedded fonts and various styles of the same font. Fonts are sorted in the `fl_GetFonts` function, eliminating deviation of cases when the same device returns fonts in the different order than in the last fingerprinting. It can be expected that the operation system settings influence the sorting order, increasing a fingerprint's entropy.

```
1  function fl_GetFonts():String
2  {
3    var allFonts:Array = Font.enumerateFonts(true);
4    allFonts.sortOn("fontName", Array.CASEINSENSITIVE);
5    var fonts:Array = new Array();
6    for (var i:Number=0; i<allFonts.length; i++) {
7      var font:Font = allFonts[i];
8      if (font.fontType != FontType.EMBEDDED && font.fontStyle ==
       FontStyle.REGULAR)
9        fonts.push(font.fontName);
10   }
11   return JSON.stringify(fonts);
12 }
```

Listing 4.5: Retrieving system fonts using the ActionScript

The `fl_GetFonts` function is called in both phases to retrieve system fonts as a JSON string. In the Mark phase the JSON string is used as an argument for a JavaScript function which sends the JSON string along with the current user identifier to the server, where retrieved fonts are stored in the database. A Recognize button's click handler calls the `fl_GetFonts` function again and pass its result to another JavaScript function, which sends the retrieved font fingerprint to the server, where it is searched for in the database and the search result is returned. The result is then shown as a dialog in the browser.

4.3.3.1 Font Fingerprinting Testing

In the analysis was stated that font enumeration can be disabled in the Adobe Flash plugin configuration, although it is enabled by default. Testing proved that all three tested browsers disable font enumeration in their anonymous mode. Therefore, font fingerprinting using Adobe Flash cannot be used to circumvent the anonymous mode in any of the tested browsers.

While the font fingerprinting depends on system fonts, it could be useful for a cross-browser tracking. The testing revealed that a retrieved font list looks the same in browsers using an NPAPI plugin, while the embedded plugin in Chrome provides a different font list. In a tested configuration, the

Table 4.10: Flash font fingerprinting testing

Browser	Works in normal mode	Can circumvent anonymous mode
Chrome	✓	
Firefox	✓	
MSIE	✓	

`Font.enumerateFonts` method returns a list of 294 fonts in Chrome, whereas the same method in Firefox and Internet Explorer browsers returns a list of 304 fonts. All retrieved fonts have the same properties – `fontStyle=regular` and `fontType=device`. Hence a fingerprint retrieved in Chrome cannot be assigned to a fingerprint retrieved in Firefox or Internet Explorer. The method's behavior is similar to the LSO supercookie.

Table 4.11: Flash font fingerprinting cross-browser testing

LSO	Obtainable from Chrome	Obtainable from Firefox	Obtainable from MSIE
Stored in Chrome	✓		
Stored in Firefox		✓	✓
Stored in MSIE		✓	✓

4.3.4 Canvas Fingerprinting

Abusing the HTML5 canvas feature for fingerprinting is different from other mentioned fingerprinting methods. While other methods extract particular pieces of information, combine them together, and use them as a fingerprint, a canvas fingerprinting result is a Base64 string depending on the user's environment properties. For that reason, only the exact match will be tested in case of canvas fingerprinting, not a similarity measure like in case of other tested fingerprinting methods.

A JavaScript `canvasFingerprint` function was used to create a canvas and fill it with different object types. The canvas object with size of 200×50 pixels was used for testing the method. It is created in document, but not placed anywhere in the DOM model, thus the user cannot see it. If the canvas were rendered to a page, it would look like at Fig. 4.1. A testing graphic contains

fonts of a different type, size, and color, special symbols, circles, gradient, and opaque elements.

Figure 4.1: Canvas fingerprinting testing graphic

The Remember phase consists of calling the `canvasFingerprint` function again and sending its result (a Base64 string) along with a user identifier to the server, where it is stored in the database. On the Recognize phase the `canvasFingerprint` function is called again, its result is sent to the server, and there it is searched for in the database. If an exact match is found, the user is recognized, otherwise not. The browser shows a dialog either with a recognized user's nickname or with a message telling the user could not be recognized.

4.3.4.1 Canvas Fingerprinting Testing

A result of canvas fingerprinting depends on the user's operation system, browser, GPU, and the GPU driver. Testing was performed on the Windows 7 operation system, running on Lenovo Z580 with following graphics adapters:

1. Intel HD Graphics 4000 (driver 10.18.10.3958)
2. NVidia GeForce GT 635M (driver 9.18.13.4520)

The first graphics adapter was selected by the operation system to use in the web browser by default. Tab. 4.12 shows that the method works well in all tested browsers and the retrieved fingerprint remains the same when running browser in the anonymous mode.

Table 4.12: Canvas fingerprinting testing

Browser	Works in normal mode	Can circumvent anonymous mode
Chrome	✓	✓
Firefox	✓	✓
MSIE	✓	✓

Testing revealed as well, that each of tested browsers provided a different fingerprint using the same GPU, its driver, and the operation system.

While the system runs the browser on the integrated graphic by default, it can be forced to run it on the dedicated one (NVidia). Testing revealed that when the browser runs on a different graphic, the fingerprint is different. Hence the testing configuration provided five different canvas fingerprints – two for each tested browser except Chrome, which could not run using NVidia graphic in tested configuration.

The canvas element provides the `getImageData` method [119], returning pixel data. The method can be used to count the difference in between collected fingerprints. Tab. 4.13 shows a different pixel count between the described test cases. Pixels are represented by their RGBA values. If any of these four parts (RGBA) differs, a pixel is considered as different. Although it is specific case results only, it looks like differences between browsers are greater than differences between graphic adapters and that Chrome browser's canvas rendering differs much from rendering in Firefox and Internet Explorer.

Table 4.13: Canvas fingerprinting pixel difference

	Chrome Intel	**Firefox Intel**	**MSIE Intel**	**Firefox NVidia**	**MSIE NVidia**
Chrome Intel	0	7038	6906	7040	6908
Firefox Intel	7038	0	3734	616	3707
MSIE Intel	6906	3734	0	3694	691
Firefox NVidia	7040	616	3694	0	3640
MSIE NVidia	6908	3707	691	3640	0

4.4 Summary

The showcase consists of ten methods – cookies, five supercookie methods, and four fingerprinting methods. All implemented methods worked in all three most common used browsers. However, it can be expected that some of these methods might fail in older browser versions, they require recent technologies not available in older browsers. This does not pose a problem, as all browser vendors seed their users with updates frequently.

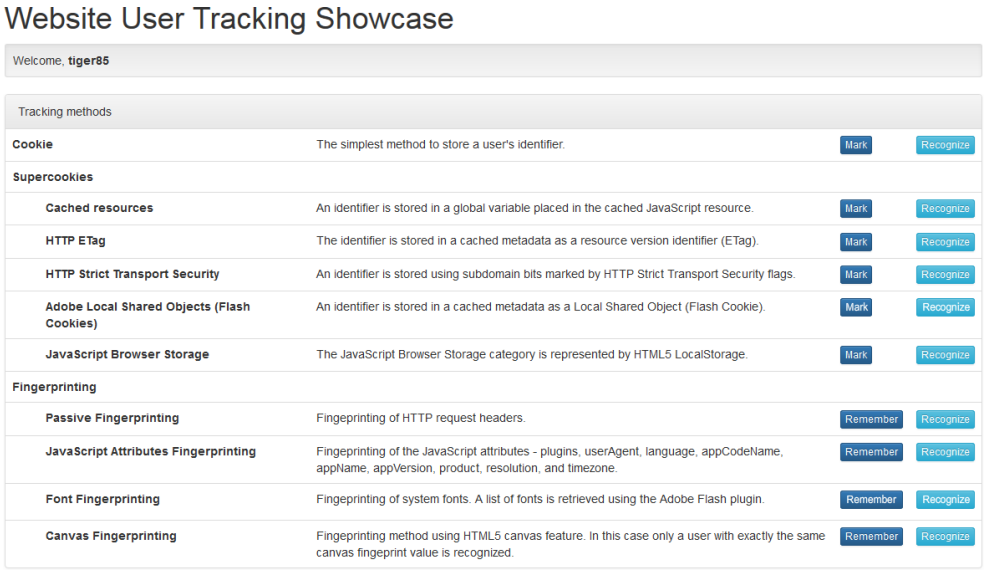

Figure 4.2: Screenshot of the implemented showcase

The analysis stated that there are many user tracking methods available, which were proven by their implementation in the showcase and by testing on the three most used browsers. Some methods were even able to circumvent the anonymous mode or identify the user even when she switched her web browser. Tab. 4.14 shows the possibilities of tested methods in three most used web browsers.

Fingerprinting methods require detailed statistics to prove their usability. The testing results in the thesis cannot prove their usability because they were performed on one device only, but another research results proved the implemented methods power in the past. On the other hand, testing results have proven possibilities of supercookies and it can be expected that they would work similarly on most users' devices.

Table 4.14: Summary of testing results

Method	Works in normal mode	Can circumvent anonymous mode	Works across browsers
Cookie	✓		
Cached content supercookie	✓		
ETag supercookie	✓		
HSTS supercookie	✓	Chrome	
Adobe LSO supercookie	✓		MSIE & Firefox
LocalStorage supercookie	✓		
Passive fingerprinting	✓	✓	
JS attributes fingerprinting	✓	✓	
Flash fonts fingerprinting	✓		MSIE & Firefox
Canvas fingerprinting	✓	✓	

Chapter 5

Conclusion

The goal of the thesis was to map methods capable of tracking website users and recognizing them on their next visits. These methods were studied and divided into three categories – cookies, supercookies, and fingerprinting. Then they were described in detail, implemented in a showcase and tested on the three most used browsers.

In the chapter 2 the history of user tracking on the Internet from the first attempts using the HTTP cookie up to modern methods abusing the modern HTTP security features like HSTS or HPKP was described. It was also stated that from the beginning of the World Wide Web the initiatives to protect against these methods achieved poor results.

In the chapter 3 there was an analysis of the tracking methods, based on the existing research. It was stated that there are plenty of places in the browser, where an identifier (supercookie) can be stored. Modern browser functions also provide much information about the user's environment allowing website to remember specific environment's details and use them to recognize the same user in the future. It was also stated that some of the described methods, like the HSTS supercookie use modern browser features and a question of browser behavior to that features is not answered in the browser documentation.

Described tracking methods were implemented in a showcase and implementation details were described in the chapter 4. In addition, the implemented methods were tested in the current version of three most used web browsers. The testing result was recorded for each method alone as well as compared together.

The thesis presented that there is an alarming amount of working user tracking methods, putting the user's privacy on the Internet in serious danger. Some of them, like the ETag supercookie, are more than 15 years old (and still working), some were invented in the last years, for instance the HSTS supercookie. The situation around the user tracking is evolving fast because there are many players in this game. On one hand, there are the website owners, advertisement companies, e-shops, analytics service providers, and social

networks. On the other hand there are browser vendors, privacy activists, and users alone. Lawmakers can influence the situation as well.

This thesis should be a warning informing about a serious threat to the user's privacy on the Internet. It presents how common well-intentioned web technologies can be abused to attack the user's privacy. The implementation and testing part shows that in most cases only a few lines of code is required to deploy a user tracking mechanism. Testing proved that most users can be tracked using the described methods, in some cases even if they use the anonymous mode or go from one web browser to another. The results confirm that tracking is a serious problem we should care about.

Bibliography

[1] The United Nations. *Declaration of Human Rights*. 1948. Available from: http://www.un.org/Overview/rights.html

[2] Internet Engineering Task Force. *RFC 7230 – Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. 2014. Available from: https://www.ietf.org/rfc/rfc7230.txt

[3] Schwarz, J. Giving the Web a Memory Cost Its Users Privacy. *The New York Times*, 4 2001. Available from: http://www.nytimes.com/2001/09/04/technology/04COOK.html

[4] Microsoft Corporation. *Microsoft Continues Commitment to Cross-Platform Solutions; Ships Microsoft Internet Explorer 2.0 for Windows 3.1*. 1996. Available from: http://news.microsoft.com/1996/04/30/microsoft-continues-commitment-to-cross-platform-solutions-ships-microsoft-internet-explorer-2-0-for-windows-3-1/

[5] MB Tech, Inc. *Microsoft Ships Internet Explorer 2.0 – This Day in Tech History*. 1995. Available from: http://thisdayintechhistory.com/11/27/microsoft-ships-internet-explorer-2-0/

[6] Berghel, H. *A Web Monopoly*. 1995. Available from: http://berghel.net/col-edit/cybernautica/jan-feb96/pcai961.php

[7] Roselli, A. *20 Years Since Netscape Navigator 1.0*. 2014. Available from: http://adrianroselli.com/2014/12/20-years-since-netscape-navigator-10.html

[8] Internet Engineering Task Force. *RFC 2109 – HTTP State Management Mechanism*. 1997. Available from: https://www.ietf.org/rfc/rfc2109.txt

[9] The Open Web Application Security Project. *Session Management Cheat Sheet*. 2015. Available from: https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

[10] Internet Engineering Task Force. *RFC 2068 – Hypertext Transfer Protocol – HTTP/1.1*. 1997. Available from: https://www.ietf.org/rfc/rfc2068.txt

[11] Ayenson, M. D.; Wambach, D. J.; Soltani, A.; et al. *Flash Cookies and Privacy II: Now with HTML5 and ETag Respawning*. 2011. Available from: http://ssrn.com/abstract=1898390

[12] World Wide Web Consortium. *Document Object Model (DOM)*. 2009. Available from: http://www.w3.org/DOM/

[13] World Wide Web Consortium. *A Short History of JavaScript*. 2012. Available from: https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript

[14] Warren, C. *The Life, Death and Rebirth of Adobe Flash*. Mashable, Inc., 2012. Available from: http://mashable.com/2012/11/19/history-of-flash

[15] Computerworld, Inc. *Stop the Flash madness – 5 bugs a week*. 2015. Available from: http://www.computerworld.com/article/2971721

[16] Soltani, A.; Canty, S.; Mayo, Q.; et al. *Flash Cookies and Privacy*. 2009. Available from: http://ssrn.com/abstract=1446862

[17] World Wide Web Consortium. *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*. 2002. Available from: http://www.w3.org/TR/P3P/

[18] Microsoft Corporation. *Platform status – Platform for Privacy Preferences 1.0 (P3P 1.0)*. 2015. Available from: https://dev.windows.com/en-us/microsoft-edge/platform/status/platformforprivacypreferences10p3p10

[19] Leon, P. G.; Cranor, L. F.; McDonald, A. M.; et al. *Token Attempt: The Misrepresentation of Website Privacy Policies through the Misuse of P3P Compact Policy Tokens*. Carnegie Mellon University, 2010. Available from: https://www.cylab.cmu.edu/files/pdfs/tech_reports/CMUCyLab10014.pdf

[20] European Union. *Cookies - European Commission*. 2015. Available from: http://ec.europa.eu/ipg/basics/legal/cookies/index_en.htm

[21] European Union. *Regulation (EC) No 45/2001 of the European Parliament and of the Council of 18 December 2000 on the protection of individuals with regard to the processing of personal data by the Community institutions and bodies and on the free movement of such data*. 2000. Available from: http://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32001R0045

[22] European Union. *Directive 2002/58/EC of the European Parliament and of the Council of 12 July 2002 concerning the processing of personal data and the protection of privacy in the electronic communications sector (Directive on privacy and electronic communications)*. 2002. Available from: http://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32002L0058

[23] European Union. *Opinion 04/2012 on Cookie Consent Exemption*. 2012. Available from: http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2012/wp194_en.pdf

[24] Center for Democracy and Technology. *Consumer Rights and Protections in the Behavioral Advertising Sector*. 2007. Available from: https://www.cdt.org/files/privacy/20071031consumerprotectionsbehavioral.pdf

[25] World Wide Web Consortium. *Tracking Preference Expression (DNT)*. 2015. Available from: http://www.w3.org/TR/tracking-dnt/

[26] Angwin, J. *Web Tool On Firefox To Deter Tracking*. The Wall Street Journal, 2011. Available from: http://www.wsj.com/articles/SB10001424052748704213404576100441609997236

[27] Albanesius, C. *Internet Explorer 10 Released for Windows 7*. PCMag Digital Group, 2012. Available from: http://www.pcmag.com/article2/0,2817,2412077,00.asp

[28] Association of National Advertisers. *ANA Board Opposes Microsoft's Decision to Implement 'Do-Not-Track' Default Function for Internet Explorer 10 Browser*. 2012. Available from: http://www.ana.net/content/show/id/analetter-microsoft#_ftn2

[29] Noyes, K. *Apache Web servers will ignore IE10's 'Do Not Track' settings*. PCWorld, 2012. Available from: http://www.pcworld.com/article/262150

[30] Microsoft Corporation. *An update on Microsoft's approach to Do Not Track*. 2015. Available from: http://blogs.microsoft.com/on-the-issues/2015/04/03/an-update-on-microsofts-approach-to-do-not-track/

[31] Eckersley, P. How Unique Is Your Web Browser? In *Privacy Enhancing Technologies*, *Lecture Notes in Computer Science*, volume 6205, edited by M. Atallah; N. Hopper, 2010, ISBN 978-3-642-14526-1, doi:10.1007/978-3-642-14527-8_1.

[32] BrowserLeaks. *BrowserLeaks.com – Web Browser Security Checklist for Identity Theft Protection*. 2015. Available from: `https://www.browserleaks.com/`

[33] Patel, L. *JavaScript/CSS Font Detector*. 2007. Available from: `http://www.lalit.org/lab/javascript-css-font-detect/`

[34] Python Software Foundation. *difflib – Helpers for computing deltas*. 2015. Available from: `https://docs.python.org/2/library/difflib.html`

[35] Internet Security Research Group. *Let's Encrypt*. 2015. Available from: `https://letsencrypt.org`

[36] Oppliger, R. *SSL and TLS: Theory and Practice*. Artech House Information Security and Privacy, Artech House, 2009, ISBN 9781596934474, 68-72 pp.

[37] The Open Web Application Security Project. *Man-in-the-middle attack*. 2015. Available from: `https://www.owasp.org/index.php/Man-in-the-middle_attack`

[38] Internet Engineering Task Force. *RFC 6797 – HTTP Strict Transport Security (HSTS)*. 2012. Available from: `https://www.ietf.org/rfc/rfc6797.txt`

[39] Marlinspike, M. *New Tricks For Defeating SSL In Practice*. Black Hat. Available from: `https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf`

[40] Internet Engineering Task Force. *RFC 7469 – Public Key Pinning Extension for HTTP*. 2015. Available from: `https://www.ietf.org/rfc/rfc7469.txt`

[41] Northcutt, C. *Rogue Certificate Authorities Put Everyone At Risk*. InterWorx LLC., 2015. Available from: `http://www.interworx.com/community/rogue-certificate-authorities-risk/`

[42] TechTarget Incorporated. *Certificate authority (CA) definition*. 2015. Available from: `http://searchsecurity.techtarget.com/definition/certificate-authority`

[43] Microsoft Corporation. *DNS defined: Domain Name System (DNS)*. 2005. Available from: https://technet.microsoft.com/en-us/library/cc787920(v=ws.10).aspx

[44] Goodin, D. *Unpatched browser weaknesses can be exploited to track millions of Web users*. Ars Technica, 2015. Available from: http://arstechnica.com/security/2015/10/unpatched-browser-weaknesses-can-be-exploited-to-track-millions-of-web-users/

[45] Internet Engineering Task Force. *RFC 6265 – HTTP State Management Mechanism*. 2011. Available from: https://www.ietf.org/rfc/rfc6265.txt

[46] Eyeo GmbH. *Adblock Plus*. 2015. Available from: https://adblockplus.org/

[47] Acar, G.; Eubank, C.; Englehardt, S.; et al. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, 2014, ISBN 978-1-4503-2957-6.

[48] Conti, G. *Googling Security: How Much Does Google Know about You?* Addison-Wesley, 2009, ISBN 9780321518668, 59–96 pp.

[49] Internet Engineering Task Force. *RFC 2965 – HTTP State Management Mechanism*. 2000. Available from: https://www.ietf.org/rfc/rfc2965.txt

[50] World Wide Web Consortium. *Web Storage (Second Edition)*. 2015. Available from: http://www.w3.org/TR/webstorage

[51] Internet Engineering Task Force. *RFC 7232 – Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests – 2.3. ETag*. 2014. Available from: https://tools.ietf.org/html/rfc7232#section-2.3

[52] Kamkar, S. *Evercookie*. 2015. Available from: https://github.com/samyk/evercookie

[53] Finley, K. *Verizon Curbs 'Zombie Cookies,' But They'll Still Stalk You*. WIRED.com, 2015. Available from: http://www.wired.com/2015/10/verizon-curbs-zombie-cookies-theyll-still-stalk/

[54] HTTP Archive. *HTTP Archive – Interesting Stats*. [from 2015-09-15]. Available from: http://www.httparchive.org/interesting.php?a=All&l=Sep%2015%202015

[55] W3Schools. *HTML Canvas Reference*. 2015. Available from: http://www.w3schools.com/tags/ref_canvas.asp

[56] Artz, D. Digital steganography: hiding data within data. *Internet Computing, IEEE*, volume 5, no. 3, 2001: pp. 75–80.

[57] Fleischer, G. *Implementing Web Tracking*. BlackHat, 2012. Available from: https://media.blackhat.com/bh-us-12/Briefings/Fleischer/BH_US_12_Fleischer_Implementing_Web_Tracking_gfleischer_WP.pdf

[58] Mayer, J. *Tracking the trackers: Microsoft advertising*. The Center for Internet and Society at Stanford Law School, 2011. Available from: http://cyberlaw.stanford.edu/blog/2011/08/tracking-trackers-microsoft-advertising

[59] StartCom Ltd. *StartSSL*. 2015. Available from: https://www.startssl.com

[60] The Chromium Projects. *HSTS Preload Submission*. 2015. Available from: https://hstspreload.appspot.com/

[61] RadicalResearch Ltd. *HSTS Super Cookies*. 2015. Available from: http://www.radicalresearch.co.uk/lab/hstssupercookies

[62] TechTarget Incorporated. *PKI (public key infrastructure) definition*. 2015. Available from: http://searchsecurity.techtarget.com/definition/PKI

[63] The Open Web Application Security Project. *Certificate and Public Key Pinning*. 2015. Available from: https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning

[64] Diquet, A. *Intercepting the App Store's Traffic on iOS*. 2013. Available from: https://nabla-c0d3.github.io/blog/2013/08/20/intercepting-the-app-stores-traffic-on-ios

[65] Internet Engineering Task Force. *RFC 5280 – Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. 2008. Available from: https://www.ietf.org/rfc/rfc5280.txt

[66] The Chromium Projects. *New Chromium security features, June 2011*. 2011. Available from: http://blog.chromium.org/2011/06/new-chromium-security-features-june.html

[67] Apple Inc. *Thoughts on Flash*. 2010. Available from: http://www.apple.com/hotnews/thoughts-on-flash/

[68] Adobe Systems Incorporated. *Flash to Focus on PC Browsing and Mobile Apps; Adobe to More Aggressively Contribute to HTML5*. 2011. Available from: http://blogs.adobe.com/conversations/2011/11/flash-focus.html

[69] Adobe Systems Incorporated. *SharedObject - Adobe ActionScript 3 (AS3) API Reference*. 2015. Available from: http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/SharedObject.html

[70] Microsoft Corporation. *ASP.NET Cookies Overview*. 2015. Available from: https://msdn.microsoft.com/en-us/library/ms178194.aspx

[71] Adobe Systems Incorporated. *Adobe Flash Player and Adobe AIR security*. 2012. Available from: http://www.adobe.com/devnet/flashplatform/whitepapers/runtime-security.html

[72] W3Schools. *Window name Property*. 2015. Available from: http://www.w3schools.com/jsref/prop_win_name.asp

[73] W3Schools. *HTML5 Local Storage*. 2015. Available from: http://www.w3schools.com/html/html5_webstorage.asp

[74] World Wide Web Consortium. *Web SQL Database*. 2015. Available from: http://www.w3.org/TR/webdatabase/

[75] Hipp, Wyrick & Company, Inc. *SQLite*. 2015. Available from: https://www.sqlite.org/

[76] World Wide Web Consortium. *Indexed Database API*. 2015. Available from: http://www.w3.org/TR/IndexedDB/

[77] Microsoft Corporation. *MSDN – userData Behavior*. 2015. Available from: https://msdn.microsoft.com/en-us/library/ms531424(VS.85).aspx

[78] Microsoft Corporation. *Microsoft Silverlight*. 2015. Available from: https://www.microsoft.com/silverlight/

[79] Microsoft Corporation. *Silverlight Isolated Storage*. 2015. Available from: https://msdn.microsoft.com/en-us/library/bdts8hk0(v=vs.95).aspx

[80] Microsoft Corporation. *Microsoft Silverlight Support Lifecycle*. 2015. Available from: https://support.microsoft.com/en-us/lifecycle?c2=12905

[81] Grossman, J. *I know where you've been.* 2006. Available from: http://jeremiahgrossman.blogspot.cz/2006/08/i-know-where-youve-been.html

[82] Stamm, S. *Plugging the CSS History Leak.* Mozilla Foundation, 2010. Available from: https://blog.mozilla.org/security/2010/03/31/plugging-the-css-history-leak/

[83] Oracle Corporation. *JNLP PersistenceService.* 2015. Available from: https://docs.oracle.com/javase/8/docs/jre/api/javaws/jnlp/javax/jnlp/PersistenceService.html

[84] Mozilla Foundation. *How to allow Java on trusted sites.* 2015. Available from: https://support.mozilla.org/en-US/kb/how-allow-java-trusted-sites

[85] Stopczynski, M.; Zugelder, M. Reducing User Tracking through Automatic Web Site State Isolations. In *Information Security*, *Lecture Notes in Computer Science*, volume 8783, edited by S. Chow; J. Camenisch; L. Hui; S. Yiu, 2014, ISBN 978-3-319-13256-3, doi:10.1007/978-3-319-13257-0_18.

[86] Center for Democracy and Technology. *Comments for November 2015 Workshop on Cross-Device Tracking.* 2015. Available from: https://cdt.org/files/2015/10/10.16.15-CDT-Cross-Device-Comments.pdf

[87] Alcorn, W.; Frichot, C.; Orru, M. *Browser Hacker's Handbook.* John Wiley & Sons, Incorporated, 2014, ISBN 9781118662090, 248-260 pp.

[88] Cover, T.; Thomas, J. *Elements of Information Theory 2nd Edition.* Wiley Series in Telecommunications and Signal Processing, Wiley-Interscience, 2006, ISBN 0471241954, 13–31 pp.

[89] StatCounter. *Browser Versions Statistics.* [from 2015-11 to 2015-12]. Available from: http://gs.statcounter.com/#all-browser_version-ww-monthly-201511-201512

[90] StatCounter. *StatCounter – Free Invisible Web Tracker, Hit Counter and Web Stats.* 2015. Available from: http://statcounter.com/

[91] Mozilla Foundation. *Fingerprinting – MozillaWiki.* 2015. Available from: https://wiki.mozilla.org/Fingerprinting

[92] Internet Engineering Task Force. *RFC 7231 – Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.* 2014. Available from: https://www.ietf.org/rfc/rfc7231.txt

[93] Herlihy, P. *How many people are missing out on JavaScript enhancement?* Government Digital Service, 2013. Available from: https://gds.blog.gov.uk/2013/10/21/how-many-people-are-missing-out-on-javascript-enhancement/

[94] InformAction. *NoScript – JavaScript/Java/Flash blocker for a safer Firefox experience!* 2015. Available from: https://noscript.net/

[95] W3Schools. *Navigator Object*. 2015. Available from: http://www.w3schools.com/jsref/obj_navigator.asp

[96] Web Hypertext Application Technology Working Group. *Navigator HW Concurrency*. 2015. Available from: https://wiki.whatwg.org/wiki/Navigator_HW_Concurrency

[97] Mozilla Foundation. *NavigatorPlugins.plugins – Web APIs*. 2015. Available from: https://developer.mozilla.org/en-US/docs/Web/API/NavigatorPlugins/plugins

[98] Mozilla Foundation. *NPAPI Plugins in Firefox*. 2015. Available from: https://blog.mozilla.org/futurereleases/2015/10/08/npapi-plugins-in-firefox/

[99] The Chromium Projects. *NPAPI deprecation: developer guide*. 2015. Available from: https://www.chromium.org/developers/npapi-deprecation

[100] W3Schools. *Screen Object*. 2015. Available from: http://www.w3schools.com/jsref/obj_screen.asp

[101] Unity Technologies. *Mobile Hardware Stats*. [from 2015-11]. Available from: http://hwstats.unity3d.com/mobile/display.html

[102] W3Schools. *Date Object*. 2015. Available from: http://www.w3schools.com/jsref/jsref_obj_date.asp

[103] W3Schools. *Navigator cookieEnabled Property*. 2015. Available from: http://www.w3schools.com/jsref/prop_nav_cookieenabled.asp

[104] W3Schools. *Navigator javaEnabled() Method*. 2015. Available from: http://www.w3schools.com/jsref/met_nav_javaenabled.asp

[105] Mulazzani, M.; Reschl, P.; Huber, M.; et al. *Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting*. Web 2.0 Security & Privacy 2013 (W2SP 2013), 2013. Available from: http://www.w2spconf.com/2013/papers/s2p1.pdf

[106] The Tor Project, Inc. *Tor Project: Anonymity Online*. 2015. Available from: https://www.torproject.org/

[107] W3Schools. *CSS font-family Property*. 2015. Available from: http://www.w3schools.com/cssref/pr_font_font-family.asp

[108] Robinson, S. *Flipping typical – compare the fonts you have*. 2009. Available from: http://flippingtypical.com/

[109] Adobe Systems Incorporated. *Font – Adobe ActionScript 3 (AS3) API Reference – enumerateFonts*. 2015. Available from: http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/text/Font.html#enumerateFonts()

[110] Rantakari, A. *Getting a List of Installed Fonts with Flash and Javascript*. 2010. Available from: http://hasseg.org/blog/post/526

[111] Adobe Systems Incorporated. *Adobe Flash Player Administration Guide for Microsoft Windows 8*. 2013. Available from: http://www.adobe.com/content/dam/Adobe/en/devnet/flashplayer/pdfs/flash_player_windows8_admin_guide.pdf

[112] Microsoft Corporation. *Fonts.SystemTypefaces Property*. 2015. Available from: https://msdn.microsoft.com/en-us/library/system.windows.media.fonts.systemtypefaces(v=vs.110).aspx

[113] Microsoft Corporation. *JavaScript API for Silverlight*. 2015. Available from: https://msdn.microsoft.com/en-us/library/cc903928(v=vs.95).aspx

[114] Oracle Corporation. *GraphicsEnvironment (Java Platform SE 7) – getAvailableFontFamilyNames*. 2015. Available from: https://docs.oracle.com/javase/7/docs/api/java/awt/GraphicsEnvironment.html#getAvailableFontFamilyNames()

[115] Oracle Corporation. *Invoking JavaScript Code From an Applet*. 2015. Available from: https://docs.oracle.com/javase/tutorial/deployment/applet/invokingJavaScriptFromApplet.html

[116] World Wide Web Consortium. *Battery Status API*. 2014. Available from: http://www.w3.org/TR/battery-status/

[117] Olejnik, L.; Acar, G.; Castelluccia, C.; et al. The leaking battery: A privacy analysis of the HTML5 Battery Status API. *IACR Cryptology ePrint Archive*, volume 2015: p. 616.

[118] Nakibly, G.; Shelef, G.; Yudilevich, S. Hardware Fingerprinting Using HTML5. *CoRR*, volume abs/1503.01408, 2015.

[119] W3Schools. *HTML canvas getImageData() Method*. 2015. Available from: http://www.w3schools.com/tags/canvas_getimagedata.asp

[120] World Wide Web Consortium. *4.11.4 The canvas element – HTML5*. 2014. Available from: http://www.w3.org/TR/html5/scripting-1.html#the-canvas-element

[121] Microsoft Corporation. *Unleash the Power of Hardware-Accelerated HTML5 Canvas*. 2015. Available from: https://msdn.microsoft.com/en-us/hh562071.aspx

[122] Mowery, K.; Shacham, H. *Pixel Perfect: Fingerprinting Canvas in HTML5*. University of California, 2012. Available from: http://cseweb.ucsd.edu/~hovav/dist/canvas.pdf

[123] Mozilla Foundation. *WebGL – Web APIs*. 2015. Available from: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

[124] Jang, D.; Jhala, R.; Lerner, S.; et al. An Empirical Study of Privacy-violating Information Flows in JavaScript Web Applications. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, 2010, ISBN 978-1-4503-0245-6.

[125] Wondracek, G.; Holz, T.; Kirda, E.; et al. A Practical Attack to De-anonymize Social Network Users. In *Security and Privacy (SP), 2010 IEEE Symposium on*, May 2010, ISSN 1081-6011, pp. 223–238, doi: 10.1109/SP.2010.21.

[126] Shaw, R. *Stealing history with CSS binary trees*. 2013. Available from: http://rileyjshaw.com/blog/stealing-history-with-CSS-binary-trees/

[127] Kotcher, R.; Pei, Y.; Jumde, P.; et al. Cross-origin pixel stealing: timing attacks using CSS filters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, CCS '13, 2013, ISBN 978-1-4503-2477-9.

[128] Zhu, Y. *Weird New Tricks for Browser Fingerprinting*. TorCon 2015, 2015. Available from: https://zyan.scripts.mit.edu/presentations/toorcon2015.pdf

[129] Grey, E. *CPU core estimation with JavaScript*. 2013. Available from: http://eligrey.com/blog/post/cpu-core-estimation-with-javascript/

[130] Kohno, T.; Broido, A.; Claffy, K. Remote physical device fingerprinting. In *Security and Privacy, 2005 IEEE Symposium on*, May 2005, ISSN 1081-6011, pp. 211–225, doi:10.1109/SP.2005.18.

[131] INRIA Rennes Bretagne-Atlantique research center. *Am I Unique?* 2015. Available from: https://amiunique.org/

[132] Pivotal Software, Inc. *Spring Framework*. 2015. Available from: http://projects.spring.io/spring-framework/

[133] The jQuery Foundation. *jQuery*. 2015. Available from: https://jquery.com/

[134] H2 Group. *H2 Database Engine*. 2015. Available from: http://www.h2database.com/html/main.html

[135] Red Hat, Inc. *WildFly*. 2015. Available from: http://wildfly.org/

[136] The Apache Software Foundation. *Maven*. 2015. Available from: https://maven.apache.org/

[137] W3Schools. *Browser Statistics*. 2015. Available from: http://www.w3schools.com/browsers/browsers_stats.asp

[138] W3Schools. *OS Platform Statistics*. 2015. Available from: http://www.w3schools.com/browsers/browsers_os.asp

[139] The Open Web Application Security Project. *Cross-site Scripting (XSS)*. 2015. Available from: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

[140] World Wide Web Consortium. *Cross-Origin Resource Sharing*. 2015. Available from: http://www.w3.org/TR/cors/

[141] Simpson, K. *Safer cross-domain Ajax with JSON-P*. Getify Solutions, 2015. Available from: http://json-p.org/

[142] Mozilla Foundation. *Shumway Project*. 2015. Available from: https://wiki.mozilla.org/Shumway

[143] Internet Engineering Task Force. *RFC 7159 — The JavaScript Object Notation (JSON) Data Interchange Format*. 2014. Available from: https://www.ietf.org/rfc/rfc7159.txt

[144] Winkler, W. E. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. In *Proceedings of the Section on Survey Research*, 1990, pp. 354–359.

[145] Oracle Corporation. *UTL_MATCH : String Matching by Testing Levels of Similarity/Difference*. 2015. Available from: https://oracle-base.com/articles/11g/utl_match-string-matching-in-oracle

[146] Q-Success. *Usage of Flash for websites*. 2015. Available from: http://w3techs.com/technologies/details/cp-flash/all/all

Appendix A

Acronyms

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

AS ActionScript

CA Certificate Authority

CORS Cross-Origin Resource Sharing

CPU Central Processing Unit

CSS Cascading Style Sheets

CSV Comma Separated Values

EC European Commission

EFF Electronic Frontier Foundation

ETag Entity Tag

ECMA European Computer Manufacturers Association

DNS Domain Name System

DNT Do Not Track

DOM Document Object Model

GPS Global Positioning System

GPU Graphics Processing Unit

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS HTTP with SSL/TLS

HPKP HTTP Public Key Pinning

HSTS HTTP Strict Transport Security

IoT Internet of Things

IP Internet Protocol

JS JavaScript

JSON JavaScript Object Notation

JSONP JSON with Padding

LSO Local Shared Object

MIME Multipurpose Internet Mail Extensions

MITM Man-in-the-middle

MSDN Microsoft Developer Network

MSIE Microsoft Internet Explorer

NPAPI Netscape Plugin Application Programming Interface

P3P Platform for Privacy Preferences Project

PKI Public Key Infrastructure

RFC Request for Comments

RGB Red, Green, Blue

RGBA Red, Green, Blue, Alpha

SPKI Subject Public Key Info

SQL Structured Query Language

SSL Secure Sockets Layer

SWF Small Web Format

TCP Transmission Control Protocol

TLS Transport Layer Security

URL Uniform Resource Locator

W3C World Wide Web Consortium

WebGL Web Graphics Library

WWW World Wide Web

XSS Cross-site Scripting

Appendix B

Showcase Manual

Prerequisites:

- JDK 8 (tested on 1.8.0_66)
- Maven (tested on 3.3.3)

No special environment setup is required to test the methods not dependent on HTTPS. The showcase can be run from a local copy of `src/showcase` folder using a single Maven command:

```
mvn wildfly:run
```

Maven will download all dependencies, build the project, download the Wildfly server environment, start the server, and deploys the showcase on it. Then the running application can be accessed on the following address:

```
http://localhost/showcase
```

In order to test the HSTS supercookie method, working HTTPS communication is required. The following steps must be done:

1. Update the hosts file (placed in the `doc` folder on the CD)
2. Import the root certification authority into the browser/operation system. The root certification authority public key is placed in the showcase folder on the CD as well.

When the hosts file is updated, the running application can be accessed on the following address:

```
http://tracking.test/showcase/
```

It is better to use the domain address than the localhost one, because of cross-site issues when calling the subdomain URLs in the HSTS supercookie method.

Detailed step by step manual for running the showcase on the clean installation of Windows 7 can be found in the `doc` folder on the CD.

Appendix C

Contents of Enclosed CD

- `CD`
  - `doc` — documentation folder
    - `hosts` — hosts file for running more subdomains locally (HSTS)
    - `manualWindows7.pdf` — step by step tutorial for running the showcase on a clean installation of Windows 7
    - `rootCA.pem` — public key of the certification authority used in the showcase
  - `src` — source code folder
    - `flash` — source code of the Flash parts of showcase
    - `showcase` — source code of the showcase (Java)
    - `thesis` — source code of the thesis