Fingerprint Privacy
A fresh perspective on web privacy

Christof Ferreira Torres
christof.ferreira.001@student.uni.lu

A thesis presented for the degree of
Bachelor in Computer Science

Local supervisor:	Dr. Ir. Hugo Jonker	hugo.jonker@uni.lu
Academic supervisor:	Prof. Dr. Sjouke Mauw	sjouke.mauw@uni.lu

Faculty of Science, Technology and Communication
University of Luxembourg
Academic Year 2013 - 2014

UNIVERSITÉ DU LUXEMBOURG

Summary

The aim of this thesis is to propose a fresh perspective on web privacy. To that end, this thesis analyses current tracking technologies such as browser fingerprinting and evaluates existing countermeasures. These countermeasures either block JavaScript respectively Flash or modify a user's browser characteristics in such a way that these are identical to the characteristics of a large set of users. However, the first approach impedes functionality and the second approach can easily be circumvented. Therefore we provide a new approach to web privacy which aims to separate so called "web identities". This separation is achieved through the creation of unique web identities for each website the user visits. This way a website cannot identify a user across different websites. We implement this approach in a Firefox extension called *Fingerprint Privacy* and test it successfully against an open source fingerprinting library. Finally we evaluate our approach and discuss future improvements.

Keywords: web privacy, fingerprinting, user tracking, detection.

Resumé

L'objectif de cette thèse est de proposer une nouvelle perspective au sujet de la protection de la vie privée sur le web. À cette fin, cette thèse analyse les technologies de traçage actuelles telles que les empreintes digitales d'un navigateur et évalue les contre-mesures existantes. Ces contre-mesures permettent soit de bloquer JavaScript, et précisément Flash, ou la modification des caractéristiques du navigateur d'un utilisateur de telle façon que celles-ci sont identiques aux caractéristiques d'un large ensemble d'utilisateurs. Toutefois, la première approche entrave les fonctionnalités et la deuxième approche est facilement contournable. Donc nous fournissons une nouvelle approche pour protéger la vie privée sur le web qui vise à la séparation de ce qu'on appelle les "identités web". Cette séparation est réalisée grâce à la création des identités web uniques pour chaque site que l'utilisateur visite. De cette manière un site web ne peut pas identifier un utilisateur sur différents sites web. Nous mettons en œuvre cette approche dans une extension pour Firefox appelé *Fingerprint Privacy* et nous la testons avec succès contre une bibliothèque open source pour les empreintes digitales. Enfin, nous évaluons notre approche et nous discutons des futures améliorations.

Mots clés: protection de la vie privée sur le web, empreintes digitales, traçage des utilisateurs, détection.

Declaration of Honesty

I hereby confirm that the present thesis is solely my own work and that if any text passages or diagrams from books, papers, the Web or other sources have been copied or in any other way used, all references – including those found in electronic media – have been acknowledged and fully cited.

Signature

Christof Ferreira Torres

Acknowledgements

I would like to thank my local supervisor Hugo Jonker for his enthusiasm, his spirit for novelty and his constructive criticism. Without his guidance and his persistent help this thesis would not have been possible.

In addition, I would like to thank Rolando Trujillo Rasúa from the University of Luxembourg and Alexandre Viejo from the Universitat Rovira i Virgili for the discussion that we had about browser fingerprinting.

Furthermore, I would like to thank Tatiana Ven for proofreading the English part, Patrick Kalenga for proofreading the French part and Laura Bock for developing the Fingerprint Privacy logo.

Finally, I would like to thank everyone else who supported me through the writing of this thesis.

Table of Contents

1 Introduction **17**

2 Web Tracking **19**
2.1 Reasons for Web Tracking . . . 19
2.1.1 Web Analytics . . . 19
2.1.2 Advertising . . . 19
2.1.3 Usability Improvement . . . 20
2.1.4 Law Enforcement . . . 20
2.2 Evolution of User Tracking . . . 20
2.2.1 Tracking Cookies . . . 20
2.2.2 Flash Cookies . . . 21
2.2.3 Evercookies . . . 21
2.2.4 Web Bugs . . . 22
2.2.5 Browser Fingerprinting . . . 22

3 Fingerprinting **23**
3.1 Browser Fingerprinting . . . 23
3.2 Browser Fingerprinting Methods . . . 24
3.2.1 Passive Browser Fingerprinting . . . 24
3.2.2 Active Browser Fingerprinting . . . 25
3.3 Robustness of Browser Fingerprints . . . 26
3.4 OSI Model Fingerprinting . . . 27

4 Related Work **29**
4.1 Panopticlick . . . 29
4.2 Real-life Implementations of Browser Fingerprinting . . . 31
4.2.1 Fingerprinting Through Browser Plugins . . . 33
4.2.2 Fingerprinting Through Unique Browser Properties . . . 34
4.2.3 Detection of Installed System Fonts . . . 34
4.2.4 Piercing Through Proxy Servers . . . 35
4.2.5 Native Fingerprinting Libraries . . . 35
4.2.6 Fingerprint Delivery Systems . . . 35
4.3 Evaluation of Existing Fingerprinting Countermeasures . . . 36
4.3.1 FireGloves . . . 36
4.3.2 Private Browsing Modes . . . 37

4.3.3 Do-Not-Track Header . . . 37
4.3.4 User-Agent Spoofing Tools . . . 38
4.3.5 Tor Browser Bundle . . . 38

5 Methodology **41**
5.1 Web Identities . . . 41
5.2 Linking Web Identities . . . 42
5.3 Maintaining Consistency between Web Identities and Browser Capabilities . . . 42
5.4 Existing Countermeasures . . . 43
5.5 Alternative Approaches to Web Privacy . . . 44
5.5.1 Hide where the Request Originated . . . 44
5.5.2 Separate Web Identities . . . 45
5.5.3 Conclusion . . . 46
5.6 Validation of the Implementation . . . 46
5.6.1 Test Case 1: First-party Fingerprinters . . . 46
5.6.2 Test Case 2: Third-party Fingerprinters . . . 47

6 Development of *Fingerprint Privacy* **49**
6.1 Structure of Firefox Extensions . . . 49
6.1.1 Chrome Folder . . . 50
6.1.2 Chrome.manifest . . . 50
6.1.3 Defaults . . . 51
6.1.4 Install.rdf . . . 51
6.1.5 Locale . . . 51
6.1.6 Modules . . . 51
6.1.7 Skin . . . 51
6.2 Packaging and Installing . . . 51
6.2.1 Packaging with Windows . . . 51
6.2.2 Packaging with Mac . . . 52
6.2.3 Packaging with Linux . . . 52
6.3 Embedding a Proxy . . . 52
6.3.1 Intercepting and Modifying HTTP requests . . . 52
6.3.2 Intercepting and Modifying HTTP responses . . . 53
6.4 Implementation of Web Identities . . . 54
6.5 Random Fingerprint Generator . . . 54
6.6 Detecting Fingerprinting Activities . . . 56
6.7 Handling Requests to Social Plugins . . . 57
6.8 Handling Requests to Browser Plugins . . . 59
6.9 User Interface . . . 59
6.9.1 Toolbar Button Popup Menu . . . 59
6.9.2 Preferences Menu . . . 60
6.9.3 Web Identities Management Menu . . . 61
6.9.4 Edit Attribute Menu . . . 62
6.10 Validation of the Extension . . . 63
6.10.1 Test Case 1: First-party Fingerprinters . . . 63

6.10.2 Test Case 2: Third-party Fingerprinters 63
6.11 Limitations of the Extension 63
6.12 Dependencies 64

7 Discussion and Future Work 65
7.1 Improving Existing Functionality 65
7.2 Adding New Functionality 66

8 Conclusion 67

Appendices 75

A List of different HTTP browser requests 77

B List of JavaScript attributes that *Fingerprint Privacy* detects and manipulates 79

C Fiches de suivi de stage 81

List of Figures

1 An example of linking web identities. 42
2 Web identities belonging to an anonymity set are still linkable. 43
3 Hiding where the request originated. 44
4 Separating web identities. 45
5 Test case 1: The inclusion of a fingerprinting library on two distinct websites. . . . 46
6 Test case 2: The inclusion of a fingerprinting library on a website through a third-party website. 47

7 An example of the hierarchical folder structure of a Firefox extension. 50
8 A representation of the workflow of our embedded proxy. 52
9 Notification banner on wwwen.uni.lu showing the list of detected attributes and the two options. 57
10 Examples of social plugins. 57
11 User interface for the toolbar button popup menu. 60
12 User interface for the preferences menu. 61
13 User interface for the web identities management menu. 62
14 User interface for the edit attribute menu. 62

List of Tables

1 User-agent strings for popular browsers. 24
2 Order of HTTP request headers for popular browsers. 25
3 Fingerprintable protocols sorted by OSI layer 27
4 Attributes and their collection method included in *Panopticlick's* fingerprints, sorted by their entropy [7]. 31
5 JavaScript-based font-probing fingerprinting scripts on the top 1M Alexa websites [2]. 32
6 Flash-based font-probing fingerprinting scripts on the top 10K Alexa websites [2]. 32
7 Comparison of all the attributes used by *Panopticlick* and the three studied fingerprinting providers [20]. 33
8 Unique navigator and screen object properties of popular browsers [20]. 34

Chapter 1

Introduction

Websites, advertising companies and social networks can track what users view while they browse the web or make a purchase online. Companies such as Google and Facebook are nowadays present on a majority of websites either through advertising services or social plugins. They can follow users all around the web and they are able to gather all sorts of information about a user such as his IP address, which websites he visits and how much time he spends on a particular website. From this gathered data can companies infer further information about a user. This causes a serious threat to privacy and web users should be concerned since this obviously shows that there are no limits to what types of information can be collected, how long it can be stored, with whom it can be shared or how it can be used or misused.

Privacy aware users want therefore to be more anonymous on the web and want to have the ability to determine what information they reveal over the Internet, who has access to their information and for what purposes their information may or may not be used. Unfortunately, it is not easy to achieve anonymity on the web. Developing countermeasures is a cat-and-mouse game where large IT companies are constantly improving their tracking techniques.

Most users these days are aware of the usage of so-called *cookies* by websites, but they do not know that these cookies also can be used by third-party websites to track users across the web and create profiles based on the websites they have visited. A lesser-known tracking technique are *evercookies* introduced by Samy Kamkar in 2010 [14]. These cookies have the particularity to be really difficult to delete.

Other widely used but still lesser-known tracking techniques are web bugs or pixel tags, these can be objects such as images embedded in websites or emails. A good example of such web bugs are *social plugins*. These social plugins are particularly concerning, for instance, the Facebook "Like" button or Twitter's "Tweet" button can collect information about which websites you visit without you having to be logged into your account [24].

A little known but powerful tracking technique is browser fingerprinting. In 2010 Peter Eckersley published a study about browser fingerprinting [7] and created a website illustrating this tracking technique, which caused a sensation and a new threat to web privacy. Browser fingerprinting techniques aim to collect as much as possible information of the browser and computer

configuration of a user in order to fully or partially identify a user across the web. The frightening part of this technique is that it is passive and that it still works even when cookies are turned off.

The application of browser fingerprinting is becoming an increasingly common practice nowadays and is being used by advertising companies and anti-fraud companies. This form of stateless user tracking allows advertising companies to bypass the limitations imposed by cookies. Moreover, with the market growth of smartphones and tablets, fingerprinting allows advertisers to augment previously gathered user-data and track the user across devices. Anti-fraud companies use fingerprinting as a method to protect users and web applications against malicious actors, for instance, by the use of stolen credentials. These services are based on massive "device reputation" databases where device fingerprints are stored along with the device owner's web history and "reputation scores".

Moreover, most current fingerprinting countermeasures such as Tor [23] and FireGloves [3], and current research approaches focus on hiding the user's real identity. We propose a radical new approach. In our view is this issue with web privacy bound to the linking of "web identities" across websites. Websites necessitate to identify users in order to provide them their services. Therefore, we generate unique web identities for individual websites and keep them separate. Since these web identities are unique, they make it impossible for websites to link them.

In this thesis we highlight existing web tracking techniques mainly focusing on browser fingerprinting. We review related work and we analyse existing countermeasures against browser fingerprinting and list their limitations. As a proof-of-concept we develop an Firefox extension enhancing the limitations of existing methods and implementing our new concept of separating web identities, in order to achieve our goal, namely to preserve privacy on the web with respect to browser fingerprinting.

Chapter 2

Web Tracking

In this chapter we discuss the context of web tracking in detail. In the first section we mention the main reasons for web tracking and we give a more clear perspective concerning the advantages and disadvantages of web tracking. In the second section we discuss the evolution of user tracking, mentioning the individual techniques currently used and their limitations which led to the development of browser fingerprinting techniques.

2.1 Reasons for Web Tracking

Web tracking is referred to the linking of visits to one or more websites as made by the same user. Web tracking can be done for numerous reasons: web analytics, advertising, usability and law enforcement. Not every form of web tracking is harmful for a user. On the one hand websites use tracking to enhance the user's browsing experience and estimate the website's performance. On the other hand websites do not miss the opportunity to collect information about their users in order to provide them more personal advertisements which increases their revenue and allows them to offer their services for free.

2.1.1 Web Analytics

Web analytic techniques focus on the number of visitors over a period of time, manners in which individuals entered the website, traffic that advertisements brings to the website, approximate geographical location of the visitors, how much time visitors spent on the website, etc. Web analytics gives websites the opportunity to evaluate their overall performance, improve their effectiveness and help managers make decisions about campaign effectiveness [15].

2.1.2 Advertising

Advertising is the main reason for tracking users because websites can earn large amounts of money by showing personalized advertisements. A website with no or only partial knowledge about a visiting user, displays randomly the advertisements which other companies paid for. As a result, website visitors watch advertisements which they are probably not interested in. For example, if men see shoe advertisements, they will ignore it whereas women will be interested in this particular product. If websites, would have more knowledge about a user, they would be

able to display personalized advertising. For this kind of advertising companies pay websites more money because it fits the user's interests which then results in increasing product selling. This form of targeted advertising can be split into contextual advertising, which is displaying advertisements connected with the content of the page a user is browsing, and behavioural advertising which categorizes the advertising content on the basis of the collected user information which may include age, gender, location, revenue, hobbies, activities and interests, etc.

2.1.3 Usability Improvement

Usability tracking brings the possibility to see where exactly users have trouble with the functionality of a website and even to categorize the computer skills of a user and adapt the page to his skills, amongst other benefits.

2.1.4 Law Enforcement

Law enforcement tracking allows to collect evidences, solve crimes, initiate investigations, find breaches of laws and identify, locate or find suspects.

However, the issue of web tracking is that it revokes the anonymity of users while they browse the web. Currently, websites try to collect all kind of information about their users. Thereby the goal is to identify a user uniquely, even if the user is not logged in or is not registered. Typically websites do not explicitly inform users about the tracking of their information. Furthermore, have web users no control about which information a website really collects about them, where it stores the information and finally who can access the information. As a result, normal web users, who are not experts in computer science, give personal information about them away, although they are not aware of it.

2.2 Evolution of User Tracking

User tacking techniques evolved over the past years due to several limitations. Moreover, each tracking technology represents an advance in the capabilities and persistence of tracking, which shows the evolution and growth of user tracking.

2.2.1 Tracking Cookies

The most basic and popular method for tracking users across the web, is to use *tracking cookies* or *third-party cookies*. These tracking cookies are normal *HTTP cookies*, also known as *web cookies*, *browser cookies* or simply *cookies*, which are supposed to outlast user sessions, hence be persistent and to store a unique identifier and return this identifier upon every HTTP request back to the server. Third-party cookies are the same as regular cookies, except that they belong to different domains than the visited website, therefore can third-party cookies track users across different websites. Nevertheless, there are several limitations concerning the usage of cookies in general:

1. Cookies are stored on a per-browser basis at the client side and as a result cookies stored on one browser are not available on another browser.

2. Cookies have a limited size. According to the RFC 6265[1], cookies should not be limited by the browsers, but browsers limit their cookies to 4096 bytes per cookie.

3. Cookies can easily be removed by the user in the browser settings.

2.2.2 Flash Cookies

Adobe Systems created the so-called *Local Shared Objects*[2] (LSOs) or also known as *Flash cookies*. These are pieces of data similar to cookies which are stored locally on a user's computer by websites using Adobe Flash[3]. LSOs are used for a variety of purposes: Flash game saves, storing site preferences and of course user tracking. LSOs have the advantage that they can be accessed across different browsers on a user's computer and therefore users can be identified behind different browsers. Another advantage that LSOs have over cookies, is the size limitation, LSOs can store up 100KB of information whereas cookies are limited by browsers to 4KB of information. Since LSOs require a browser to have the Flash plugin installed, the plugin itself is responsible to manage the storing of the LSOs. As a result of this, browsers can not easily delete these LSOs. However, as of January 5th 2011[4], Adobe Systems, Google Inc., and the Mozilla Foundation, published a new browser API named "NPAPI ClearSiteData"[5]. This allows browsers implementing the API to treat LSOs the same way as HTTP cookies and thus deletion rules that previously applied only to HTTP cookies are now also applied to LSOs.

2.2.3 Evercookies

Regular cookies are easy to delete and for this reason Samy Kamkar created in 2010 *Evercookies* or also known as *Zombie cookies* or *Supercookies*. These cookies can be recreated after deletion from information stored outside of the browsers dedicated storage. This information is stored locally on the user's computer on various locations: [14] as standard HTTP cookies, LSOs, Silverlight Isolated Storage, RGB values of auto-generated, force-cached PNGs using HTML5 Canvas tag to read pixels back out, web history, HTTP ETags, web cache, window.name caching, Internet Explorer userData storage, HTML5 Session Storage, HTML5 Local Storage, HTML5 Global Storage, HTML5 Database Storage via SQLite. The information is stored on various locations, in order to assure the recreation of the cookie. Some of these storage locations can be read from any browser and thus a user can be identified across different browsers. Consequently, these cookies are very hard to remove and can be cross-browser compatible due to LSOs and HTML5 storage mechanisms.

Although, since the 26th May 2012 the EU e-Privacy Directive, or also denoted as the EU Cookie Law, enforces websites to get their visitors informed consent before placing a cookie on their computer [21]. This law does not only apply for cookies but it also affects anything that acts like a cookie, for example, Flash cookies and HTML5 Local Storage. However, many websites still ignore this law because there are no real prosecutions being done and because it is more a directive

[1]HTTP Cookie, RFC 6265 – http://tools.ietf.org/html/rfc6265
[2]Adobe Local Shared Objects – http://www.adobe.com/security/flashplayer/articles/lso/
[3]Adobe Flash – https://www.adobe.com/software/flash/about/
[4]NPAPI Clear plugin site data – https://mail.mozilla.org/pipermail/plugin-futures/2011-January/000298.html
[5]NPAPI:ClearSiteData – https://wiki.mozilla.org/NPAPI:ClearPrivacyData

of the EU and therefore the EU member states still have the last word regarding if they enforce this law or not.

2.2.4 Web Bugs

Web bugs or also known as *web beacons* or *pixel tags*, are objects embedded in websites or emails. Mostly web bugs are a 1x1 image which is included in a piece of HTML code which is usually invisible to the user due to the fact of being so small. They are used to track when and from which device a user is reading a website or email. When a user accesses a website or opens an email, all of the images are loaded. For each image a request is sent out and the website receiving the request knows where the request originated. Social networking websites are particularly using extensively these web bugs, since it's interesting for them knowing on which websites their users go to create profiles and provide their users personalised advertising. A good example of such web bugs are Facebook's like button [9], Twitter's tweet button [13] or Google's +1 button [10]. Almost every website nowadays offers these social buttons. What makes these buttons so terrifying, is that for instance, even if you don't click on the like button, Facebook knows what website you visited. When your browser visits the website, it makes a request to Facebook asking for the image of the like button and sends at the same time a cookie identifying you.

2.2.5 Browser Fingerprinting

All the above mentioned user tracking techniques depend on some information stored on the user's computer and thus are vulnerable to deletion. Browser fingerprinting is a passive tracking technique and therefore more robust against manipulation by the user. Consequently, future user tracking techniques are not focusing anymore on the authentication factor of what the user "has" but instead they are focusing on the authentication factor of what the user "is", in order to identify a user. It's harder to hide what you are than what you have. Since this is a new way of identification we do not know if existing privacy enhancing techniques are effective.

Chapter 3

Fingerprinting

Fingerprinting or device fingerprinting can be seen as a form of a side channel attack. Many kinds of electronic devices leak unintentionally subtle but measurable information which allows them to be "fingerprinted". These devices can be entirely or partially identified through processing of their outputs or communications [16]. A device fingerprint or browser fingerprint is the combination of information collected about a remote computing device for the purpose of identification.

In the first section of this chapter we outline the definition and characteristics of browser fingerprinting. In the second section we mention possible browser fingerprinting methods for the extraction of fingerprintable information, whereas we divided them into passive and active fingerprinting methods. The third section we list the current robustness of browser fingerprints. In the final section we discuss fingerprinting from the perspective of the OSI model.

3.1 Browser Fingerprinting

Browser fingerprinting is the process of collecting sufficient information through the web browser such that a website is capable to perform a stateless device identification in order to identify or re-identify a visiting user. The combination of this information, also denoted as browser fingerprint, is then used as identifier for tracking the user across the web. Naturally, when a user browses to a website that includes fingerprinting technology and he logs in, then his browser fingerprint may be collected and associated with his user profile and compared to a database of known browser fingerprints. Using such techniques, known browser fingerprints can be matched, and previously unknown browser fingerprints can be added to the database.

There are several reasons why websites may need device specific information, e.g. to correctly render content or serve device compatible media. As a result, there are many APIs available in different programming languages which enable websites to query for these attributes [2]. If all of these attributes are taken separately, then they don't provide enough information to identify a device uniquely. Combined, this information is far more revealing.

Fingerprints can be used as global identifiers, if there is enough entropy in the distribution of a given fingerprinting algorithm to make a subset of users uniquely recognizable. Fingerprints can also be used in combination with IP addresses as cookie regenerators. A significant number

of websites may use evercookies or supercookies as a way to regenerate normal cookies that the user has previously deleted. Fingerprints may act as successors for evercookies and supercookies in order to regenerate cookies [7]. In particular, a fingerprint that carries no more than 15-20 bits of entropy might be in almost all of the cases sufficient to uniquely identify a particular browser, given its IP address. Therefore, a final usage for fingerprints might be the combination of fingerprint and IP address in the absence of cookies, hence as a means of distinguishing multiple users behind a single IP address, even if those users block cookies entirely [27].

3.2 Browser Fingerprinting Methods

In general, browser fingerprinting methods can be divided into *passive* and *active* methods. Clearly nothing prohibits the combination of passive and active fingerprinting methods which then results in a lot more effective fingerprinting methods.

3.2.1 Passive Browser Fingerprinting

Passive browser fingerprinting methods are based on characteristics observable in the contents of web requests, without the use of any code executing on the client side. Passive fingerprinting can therefore include cookies, HTTP request headers, IP address and other networking related information. The user-agent string, for example, is an HTTP request header that typically identifies the browser, rendering engine, browser version and operating system. For some cases, the combination of the user-agent string and the IP address will commonly uniquely identify a particular user's browser. This is due to the fact that the user-agent string reveals a lot of unnecessary unique information, for example, long browser and operation system version numbers. In Table 1 we can recognize that browsers based on Apple's WebKit rendering engine i.e. Safari, Opera and Chrome are significantly affected by this issue. These browsers reveal the minor version of the operating system and the minor version of the rendering engine, which as a result makes them more fingerprintable.

Browser	**User-Agent**
Mozilla Firefox	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:29.0) Gecko/20100101 Firefox/29.0
Apple Safari	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.75.14 (KHTML, like Gecko) Version/7.0.3 Safari/537.75.14
Opera	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/34.0.1847.132 Safari/537.36 OPR/21.0.1432.57
Google Chrome	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/34.0.1847.131 Safari/537.36
Microsoft Internet Explorer	Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko

Table 1: User-agent strings for popular browsers.

The order in which browsers send the HTTP headers and the variation in HTTP accept headers across requests for different content types[1], is a further example of a passive fingerprinting method. In Table 2 we can see an example of the order of HTTP request headers to the website "uni.lu" made by different popular browsers (see Appendix A for a more detailed list). We perceive that the order in which a browser sends the HTTP headers depends of the rendering engine it uses. Mozilla Firefox uses the rendering engine Gecko and therefore sends after the GET request the host, then the user-agent and finally the accept, accept-language and the accept-encoding header. Furthermore, we state that Apple Safari, Opera and Google Chrome send the HTTP headers in the same manner by sending after the GET request, the host, then the accept header, then the user-agent and finally the accept-encoding and the accept-language header. This is due to the fact that those three browsers are based on Apple's WebKit rendering engine. As a last observation, Microsoft's Internet Explorer uses it's own rendering engine named Trident, which sends the HTTP headers in a totally different order then the previous browsers did. It sends after the GET request first the accept and the accept-language header, then the user-agent, then the accept-encoding header and finally the host. Unfortunately, there is no given standard which implies rendering engines to send the HTTP headers in a defined way. This is bad for privacy, since the order reveals additional distinguishable information and because a website might still find out the true nature of a user's browser with the help of the order of the HTTP headers, even if a user is spoofing his user-agent.

Mozilla Firefox	**Apple Safari, Opera, Google Chrome**	**Microsoft Internet Explorer**
Host	Host	Accept
User-Agent	Accept	Accept-Language
Accept	User-Agent	User-Agent
Accept-Language	Accept-Language	Accept-Encoding
Accept-Encoding	Accept-Encoding	Host

Table 2: Order of HTTP request headers for popular browsers.

3.2.2 Active Browser Fingerprinting

Active browser fingerprinting methods are techniques where a website runs some JavaScript or other code on the local client computer in order to observe additional characteristics about the browser or computer. Techniques for active fingerprinting might include accessing the screen size, enumerating fonts or plugins, evaluating performance characteristics or rendering graphical patterns. Examples of such active methods are:

- Collecting device identifiable information (e.g. CPU speed, CPU manufacturer, installed programs, etc) through plugin APIs such as Microsoft's Silverlight API, Adobe Systems Flash API or Oracle's Java API.
- Clock skew measurements, thus measuring the time difference between a user's computer and a server, where 41st Parameter [1] is claiming to look at more than 100 parameters and at the core of its algorithm being such a time differential parameter.

[1] http://www.newmediacampaigns.com/blog/browser-rest-http-accept-headers

- TCP stack fingerprinting, thus the detection of irregularities in the TCP/IP stack and the piercing through proxy servers in order to retrieve the clients real IP address, where ThreatMetrix [26] is claiming to be able to do that.

- The detection of visited websites through the CSS history detection hack [8]. It detects from a list of websites, which websites a user has or has not visited.

- The detection of installed system fonts by using Flash, Java or any other plugin. However, there is also a way of detecting the installed system fonts by using JavaScript and CSS [22].

- A wide range of subtle JavaScript behavioural tests, that can be used to measure the performance signature of a browser's JavaScript engine, allowing the detection of browser version, operating system and micro architecture [17,19]. Another example is to subvert the whitelist mechanism of the popular NoScript Firefox extension through a website determining if particular domains exist in a user's NoScript whitelist [17].

- HTML5 Canvas fingerprinting, where the same HTML5 Canvas element can produce different pixels on a different browser, depending on the system on which it is executed [18].

It should be noted that the above listed active fingerprinting methods are only the tip of the iceberg and that there are potentially more methods which help to collect useful information to include in fingerprints.

3.3 Robustness of Browser Fingerprints

Numerous events can cause a browser fingerprint to change. These events may include browser updates, installing or upgrading plugins, disabling cookies or JavaScript, installing a new font or installing an external application which includes new fonts, buying a new monitor or in cases of a laptop connecting an external monitor which then alters the screen resolution. However, the robustness of browser fingerprints can be increased due to the fact that there are some values that remain constant or rarely change after the installation of a system. Browser, plugin and font upgrades can be detected and matched to previous fingerprints with the help of IP addressees or with the help of fingerprints contained in a database having a high matching score. Dynamic IP address changes can be solved by using only the two first octets of an IP address [4]. Eckersley et al. [7] implemented a very simple algorithm to heuristically estimate whether a given fingerprint might be an evolved version of a fingerprint seen previously. Their heuristic made a correct guess in 65% of the cases, an incorrect guess in 0.56% of the cases and no guess in 35% of the cases. Incredible 99.1% of the guesses were correct, while the false positive rate was only 0.86%. They also state that their algorithm is very crude and that with no doubt it could be significantly improved with some effort. Although fingerprints turn out not to be particularly stable, browsers reveal so much version information and configuration information that they remain overwhelmingly trackable.

3.4 OSI Model Fingerprinting

Fingerprinting can also be seen from a more general perspective, for example with respect to the OSI model. The OSI model is a conceptual model that characterizes and standardizes the internal functions of a communication system by partitioning it into seven abstraction layers. Most fingerprinting algorithms operate only on the top layer, the application layer, because it is the closest to the user, the user mainly interacts with this layer. Another reason for fingerprinting algorithms to operate on the top layer, is the convenience to program whereas lower layers would include a larger complexity which would require for instance to reimplement the TCP/IP stack in order to retrieve more detailed information about the network and system. As an example, various network protocols transmit or broadcast packets or headers from which one may infer client information. Examples of such protocols are: FTP, HTTP, Telnet, DHCP, TLS/SSL, SNMP, NetBIOS, TCP, IPv4, IPv6, ICMP, SMB, CDP, IEEE 802.11, etc. Table 3 illustrates some of these protocols sorted by layer. However, OSI model fingerprints can operate on more than one layer and thus go beyond the scope of simple browser fingerprinting.

OSI layer	**Protocol**
Layer 7: Application layer	FTP, HTTP, Telnet, DHCP
Layer 6: Presentation layer	TLS/SSL
Layer 5: Session layer	SNMP, NetBIOS
Layer 4: Transport layer	TCP
Layer 3: Network layer	IPv4, IPv6, ICMP
Layer 2: Data link layer	SMB, CDP
Layer 1: Physical layer	IEEE 802.11

Table 3: Fingerprintable protocols sorted by OSI layer

Chapter 4

Related Work

In this chapter we review related work discovered through our research whereas the limitations we came across later also conducted to the development of certain functionalities of our Firefox extension. In the first section we discuss the *Panopticlick* fingerprinting algorithm from P. Eckersley [7] and list some of his major observations. In the second section we consider commercial fingerprinting, the real-life implementations of fingerprinting libraries discovered by G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens & B. Preneel [2] through their self-developed framework *FPDetective*. We also consider a later study by N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kreugel, F. Piessens & G. Vigna [20] which attempts to examine the problem of browser fingerprinting from all of the players involved, i.e. from the perspective of the fingerprinting providers and their fingerprinting methods, the sites making use of fingerprinting and the users who employ privacy-preserving extensions to fight fingerprinting. In the third section we discuss privacy-preserving extensions, where we debate about *FireGloves*, an Firefox extension developed by K. Boda, Á. M. Földes, G. Gulyás & S. Imre in 2011 [4] as a response to Eckersley's study, and other existing fingerprinting countermeasures. We briefly explain their core functionalities and highlight their current limitations.

4.1 Panopticlick

In 2010, P. Eckersley from the Electronic Frontier Foundation earned several merits for conducting the first large-scale study concerning browser fingerprinting [7], and detailing one of the various possible methods of fingerprinting a browser. For demonstration purposes he created a website named *Panopticlick*[1], and illustrated how unique browser fingerprints could help websites to identify individual users across the web without the need of client-side storage technologies, such as cookies. The website tests the uniqueness of your browser based on the information it shares with the website i.e. trough reading the HTTP headers and with the help of JavaScript, Flash or Java. Surprisingly, according to his results is this method extremely precise. 94.2% of the visiting browsers with Flash or Java enabled could be uniquely identified.

Eckersley describes one possible implementation of a fingerprinting algorithm by collecting a number of commonly known and less-commonly known characteristics that modern browsers make

[1] Panopticlick – https://panopticlick.eff.org/

available to websites. A small part of these characteristics can be inferred from the HTTP requests sent by the browser to the website. However, the majority of these characteristics are collected through JavaScript or Flash and are asynchronously (e.g. AJAX) sent back to the website. The collected data is grouped into eight separate attributes, listed in Table 4, though some of these attributes comprise multiple related details. The fingerprint is essentially the combination of these attributes.

The entropy measures how much identifying information is revealed. This can be expressed in bits whereas an entropy of 1 bit denotes 2 possible values, an entropy of 2 bits denotes 4 possible values, etc. Adding one more bit of entropy doubles the number of possibilities. For instance, there are around 7 billion humans on the planet, the identity of a random, unknown person contains just under 33 bits of entropy ($2^{33} \approx 8$ billion), $\Delta S = log_2(\frac{1}{6625000000}) = 32.6$ bits of information. When we learn a new fact about a person, that fact reduces the entropy of their identity by a certain amount:

$$\Delta S = -log_2(P(X = x))$$

Where ΔS is the reduction in entropy, measured in bits, therefore 2 as the logarithm base, and $P(X = x)$ is simply the probability that the fact would be true of a random person. As an example:

Starsign: $\Delta S = -log_2 \cdot P(\text{STARSIGN} = \text{lion}) = -log_2(\frac{1}{12}) = 3.58$ bits
Birthday: $\Delta S = -log_2 \cdot P(\text{BIRTHDAY} = \text{20th of August}) = -log_2(\frac{1}{365}) = 8.5$ bits

However, the combining of information depends on whether the information is independent. For instance, if you know someone's birthday and gender, you have $8.51 + 1 = 9.51$ bits of information about their identity because the probability distributions of birthday and gender are independent. But the same is not true for birthdays and starsigns. If you know someone's birthday, then you already know their starsign, and being told their starsign doesn't increase the information at all, $8.51 + 3.58 = 8.51$ bits of information. We want to calculate the change in conditional entropy of the person's identity on all the observed variables. We can do that by making the probabilities for new facts conditional on all the facts we already know [6].

Therefore, the amount of revealing information in a specific field depends on how much we already know about this field. This combination can again be expressed in bits by Shannon's entropy formula [25]. Whereas for n different variables $X_1, X_2, ..., X_n$ with probabilities $p_1, p_2, ..., p_n$, the total amount of revealing information of the field is specified by the following formula:

$$H = -\sum_{i=1}^{n} p_i \cdot log_2(\frac{1}{p_i})$$

In Table 4 we can observe that the list of installed browser plugins, the list of installed system fonts and the user-agent, are the three attributes which return the highest entropy, thus provide the most information. The list of browser plugins provides 15.4 bits of entropy, the list of system fonts provides 13.9 bits of entropy and the user-agent provides 10.0 bits of entropy. The list of browser plugins and system fonts reveal a large amount of information because of their order.

Unfortunately the API's do not return the list of browser plugins or system fonts sorted back and if a browser plugin or system font is newly installed or updated, then it is moved to the top of the list. Therefore, is the order different from operating system to operating system and even from computer to computer. Furthermore, provide browser plugins more information than system fonts because they return the exact plugin version numbers e.g. Java 1.6.0_17 or DivX Web Player 1.4.0.233.

Attribute	**Entropy (bits)**	**Collectable via**			
		HTTP	**JavaScript**	**Flash**	**Java**
Browser plugin list (plugin versions and Mime-types)	15.4		✓		
System font list	13.9		✓	✓	✓
User-Agent	10.0	✓	✓		
HTTP Accept headers	6.09	✓			
Screen resolution	4.83		✓		
Timezone	3.04		✓		
Partial supercookie test	2.12		✓		
Cookies enabled	0.353	✓			

Table 4: Attributes and their collection method included in *Panopticlick's* fingerprints, sorted by their entropy [7].

4.2 Real-life Implementations of Browser Fingerprinting

In 2013, G. Acar and his working colleagues from KU Leuven reported in their paper [2] the implementation and the deployment of a framework called *FPDetective*, for the detection and analysis of web-based fingerprinters. Their framework is based on a crawler, whose purpose is to visit websites and collect data about events that might be related to fingerprinting, such as the loading of system fonts or the reading of specific browser properties. The collected data is then parsed and committed to a central database. To detect Flash-based fingerprinting, all crawler traffic is directed to an intercepting proxy, where the network dumps are parsed to extract Flash objects. These Flash objects are then decompiled using a free third-party decompiler and stored in the central database. By applying their framework, with focus on font detection practices, the researchers were able to conduct a large-scale analysis of the top million most popular websites on the internet given by the Alexa global ranking[2]. They discovered that the adoption of fingerprinting is much higher than previous studies had estimated. FPDetective found 13 instances of JavaScript-based font-probing scripts, on a total of 404 websites. Table 5 lists these 13 scripts sorted by the number of websites using the particular script.

[2]Alexa top ranking – http://www.alexa.com/topsites

Fingerprinting provider	Script name	Number of websites
BlueCava	BCAC5.js	250
Perferencement	tagv22.pkmin.js	51
CoinBase	application-773a[...snipped...].js	28
MaxMind	device.js	24
Inside graph	ig.js	18
SiteBlackBox	No fixed URL	14
Analytics-engine	fp.js	6
Myfreecams	o-mfccore.js	3
Mindshare Tech.	pomegranate.js	3
Cdn.net	cc.js	3
AFK Media	fingerprint.js	2
Anonymizer	fontdetect.js	1
Analyticsengine	fingerprint.compiled.js	1

Table 5: JavaScript-based font-probing fingerprinting scripts on the top 1M Alexa websites [2].

In addition to the 13 JavaScript-based font-probing scripts, FPDetective found 6 Flash-based font-probing scripts on a total of 95 websites of the top 10 thousand Alexa global ranking websites. Table 6 lists these 6 scripts sorted by the number of websites using the particular script.

Fingerprinting provider	Script name	Number of websites
BB Elements	bbnaut.swf	69
Piano Media	novosense.swf	12
BlueCava	guids[2-3].swf	6
ThreatMetrix	fp.swf	6
Alipay	lsa.swf	1
MEB (Turkish Ministry of Education)	502758.swf	1

Table 6: Flash-based font-probing fingerprinting scripts on the top 10K Alexa websites [2].

A few months later, Nick Nikiforakis and his colleagues from KU Leuven took this research a step further. They analysed in detail [20] the code of three popular browser fingerprinting code providers and compared them to the code of Panopticlick. These companies are BlueCava[3], Iovation[4] and ThreatMetrix[5]. In Table 7 we can compare the attributes used by Panopticlick with the attributes used by these three companies. We can state that these companies share some attributes with Panopticlick, but that they also included new attributes in their fingerprinting solutions not used by Panopticlick.

[3]BlueCava – http://bluecava.com/
[4]Iovation – https://www.iovation.com/
[5]ThreatMetrix – http://www.threatmetrix.com/

Attribute	Panopticlick	Fingerprinting provider		
		BlueCava	Iovation	ThreatMetrix
Plugin enumeration	✓	✓		✓
Mime-type enumeration	✓			✓
ActiveX + CLSIDs	✓	✓		✓
Google Gears Detection		✓		
Flash Manufacturer				✓
Cookies enabled	✓			
Timezone	✓	✓	✓	✓
Flash enabled	✓	✓	✓	✓
Browser Language		✓	✓	✓
System/User Language		✓		
Do-Not-Track User Choice		✓		
MSIE Security Policy		✓		
Date & time			✓	
Proxy Detection			✓	✓
User-Agent	✓	✓	✓	✓
HTTP Accept headers	✓			
Partial supercookie test	✓			
Math constants		✓		
AJAX Implementation		✓		
Font Detection	✓	✓		✓
Windows Registry		✓	✓	
MSIE Product key			✓	
OS + Kernel version				✓
Screen Resolution	✓	✓	✓	✓
Device Enumeration		✓		
Device Identifiers			✓	
IP address		✓		
TCP/IP Parameters		✓	✓	

Table 7: Comparison of all the attributes used by *Panopticlick* and the three studied fingerprinting providers [20].

4.2.1 Fingerprinting Through Browser Plugins

All three companies use Flash in addition to JavaScript to fingerprint a user's environment, whereas Panopticlick only uses Flash to retrieve the list of system fonts. Surprisingly, none of these companies uses Java, whereas one does have dead Java code in their library, but does not make any use of it. This is most likely due to the low market penetration in browsers and the fact that a user gets notified and must give permission to a website when it is trying to run a Java applet. However, Panopticlick does uses Java in order to retrieve the list of installed system fonts when Flash and Javascript are disabled. A reason why these companies use Flash is because it is transparent to the user, the API is richer in functionality and provides more detailed information

as the JavaScript API. For instance, a Linux user running Firefox on a 64-bit machine will with JavaScript report about the platform "Linux x86_64". Flash on the other hand, will report about the platform the full kernel version e.g. "Linux 3.2.0-26-generic". This additional information is not only unwelcome concerning privacy, but in addition this is also bad from a security perspective, since an attacker knows not only the browser and the architecture but also the specific kernel version of the system and therefore he could launch a tailored attack on the system. Another API call that behaves peculiarly is the reporting of the screen resolution. When a user utilizes a dual-monitor setup, Flash reports as the width of the screen the sum of the two individual screens. This value, when combined with the JavaScript API call, which lists the resolution of the monitor where the browser window is located, allows the detection of a multiple-monitor setup.

4.2.2 Fingerprinting Through Unique Browser Properties

Another significant difference from Panopticlick is that these three companies are not trying to operate in the same way across all browsers, but given a specific browser, they try to read unique properties provided by this browser. For example, when Internet Explorer is detected, then they try to extensively fingerprint Internet Explorer specific properties, such as `navigator.securityPolicy` and `navigator.systemLanguage`. See Table 8 for a more detailed list of unique properties of the `navigator` and `screen` object of popular browsers. This information can be not only used to fingerprint a user but also detect a fake user-agent.

Browser	**Unique properties**
Mozilla Firefox	screen.mozBrightness, screen.mozEnabled, navigator.mozSms, +10
Opera	navigator.browserLanguage, navigator.getUserMedia
Google Chrome	navigator.webkitStartActivity, navigator.getStorageUpdates
Microsoft Internet Explorer	screen.logicalXDPI, screen.fontSmoothingEnabled, navigator.appMinorVersion, +11

Table 8: Unique navigator and screen object properties of popular browsers [20].

4.2.3 Detection of Installed System Fonts

Nikiforakis et al. [20] validate what Eckersley [7] already denoted. The list of installed system fonts can serve as an important part of a user's unique fingerprint owing to the fact that it provides a large entropy. Since JavaScript does not provide a direct way of retrieving this list, all three companies use plugin-based detection to gather the list of installed system fonts. Surprisingly, only one company is preserving the order of the list of installed system fonts, whereas the other companies probably are unaware of the fact that the order of fonts, likely to the order of installed plugins, is stable and machine specific and thus can be used as an additional fingerprinting characteristic. However, one company also uses JavaScript as a fall-back method for font-detection. This is more a side-channel inference, where they make use of a technique similar to the JavaScript/CSS Font Detector [22], in order to identify the presence or absence of any given font from a list of fonts.

The downside of this approach is that less popular fonts are not detected and the order of the fonts is no longer a fingerprintable characteristic.

4.2.4 Piercing Through Proxy Servers

A specific IP address can be an important feature for fingerprinting and assuming that anti-fraud solutions use fingerprinting for the detection of fraudulent activities. The distinction between a user who is situated in a specific country or a user who pretends to be in a specific country, is crucial. Therefore, are two of the three companies trying to detect the user's real IP address, or at least detect if the user is making use of a proxy server. The companies use Flash to retrieve the user's real IP address. Flash has the ability to circumvent the user-set proxies at the level of the browser and thus directly contact a remote host, disregarding any browser-set HTTP proxies. To detect the present of a proxy server, the companies exchange identifiers between JavaScript and Flash. These identifiers are used to correlate two possibly different IP addresses. If a JavaScript originating request has the same identifier as a Flash originating request from a different IP address, then the server can be certain that the user is making use of an HTTP proxy.

4.2.5 Native Fingerprinting Libraries

Previous browser fingerprinting solutions and research focused on utilizing as much as possible of the API surface of existing popular browser plugins in order to obtain user-specific data. However, two out of the three companies probe a user's browser for a special plugin. This special plugin is essentially a native fingerprinting library, which is distributed as a CAB file for Internet Explorer and eventually loads as a DLL inside the browser. DLL's are able to retrieve system-specific attributes, such as the hard disk's identifier, TCP/IP parameters, the computer's name, Internet Explorer's product identifier, the installation date of Windows, the Windows Digital Product Id and the installed system drivers. All of these attributes combined provide a much stronger fingerprint than what JavaScript or Flash could ever construct. It is also worthwhile mentioning that one of the two plugins was misleadingly identifying itself as "ReputationShield" when asking the user for permission and none of the 44 antivirus engines of VirusTotal identified the fingerprinting DLL's as malicious.

4.2.6 Fingerprint Delivery Systems

Nikiforakis et al. [20] made in addition to the previous mentioned discoveries, an even more petrifying discovery on how commercial fingerprinting providers deliver their services. In the experiments of Eckerlsley [7] was a 1-to-1 relationship between the page creating the fingerprint and the backend storing the fingerprint. However, in commercial fingerprinting, there is a N-to-1 relationship because each company provides his fingerprinting services to many websites and needs to get back the user fingerprints from each of these websites. This was not determined by chance but to the contrary, there is a smart business model behind it. This model allows commercial fingerprinting providers to provide and improve their services at the same time.

They discovered two different scenarios for fingerprinting. For the first scenario, the first-party website was not directly involved in the fingerprinting process. Instead, the fingerprinting code

was delivered by an advertising network which in turn sent back the resulting fingerprint to the fingerprinting company. They assume that this was most likely done by the advertising network in order to fight click-fraud and that it is rather unclear if the first-party website knows that its users are being fingerprinted. For the second scenario, the first-party website directly includes the fingerprinting library. The fingerprinting companies BlueCava and Iovation combine all features into a single fingerprint, the fingerprint is DES-encrypted, concatenated with the encrypted keys and finally converted to a Base64 encoding. The resulting string is then added as a new hidden input element in the first-party website's login form. This way the fingerprint is saved to the first-party's server when the user logs in. However, the first-party website cannot decrypt the fingerprint, since the DES keys were generated on the fly and then encrypted with a public key. The first-party website must therefore submit it back to the fingerprinting company. The fingerprinting company then answers with a reputation score and other device information respectively user information. This architecture allows BlueCava and Iovation to hide the implementation details from their clients and to correlate user profiles across its entire client-base. ThreatMetrix works in a different way. The first-party website must first create a session identifier and place it into an `<div>` element with a predefined id. The script of ThreatMetrix reads this session identifier and appends it to all requests towards its server. This means that the first-party website never gets a user's fingerprint but only receives information about the user by querying ThreatMetrix for specific session identifiers.

4.3 Evaluation of Existing Fingerprinting Countermeasures

In the previous section it has been shown that most of the fingerprintable information is provided through JavaScript and Flash, so disabling these would significantly decrease the information provided and therefore the ability to track browsers. However, this would also mean a significant decrease in the browsing experience, since nowadays many web technologies and websites are based on JavaScript and Flash. In this section we will mention the main functionalities and the limitations of some currently existing countermeasures against fingerprinting, such as the FireGloves Firefox extension, the private browsing modes of browsers, the Do-Not-Track header, some user-agent header spoofing tools and finally the Tor Browser Bundle.

4.3.1 FireGloves

FireGloves [3] is a proof-of-concept browser extension for Firefox developed by K. Boda et al. [4] for research purposes. FireGloves returns user-defined or random values from a defined list, when queried for certain attributes such as the screen resolution, the platform on which the browser is running, the browser's vendor, the browser's version, and the browser language. Although FireGloves tries to randomly spoof some attributes of the browser, it always spoofs the browser's user-agent and platform in the same manner i.e. pretending to be Mozilla Firefox version 6 running on Windows. This is because K. Boda et al. aim to make the user being part of a majority of users and this configuration came out to be, at the time of their research, the most common configuration. However, this configuration is not the most commonly used anymore and the extension does not take fully care of consistency, for example it does not modify the `navigator.oscpu` object and it does not remove any of the new unique properties introduced in later versions of

Mozilla Firefox, such as `navigator.mozCameras` and `navigator.doNotTrack`. Additionally, FireGloves limits the number of fonts that a single browser tab can load and reports false dimension values for the `offsetWidth` and `offsetHeight` properties to make JavaScript-based font detection useless. However, due to the limitation of the number of fonts, some websites may be displayed in an unpleased and unintended manner for the user and therefore have an impact on user experience. Another shortcoming, instead of relying on the `offsetWidth` and `offsetHeight` properties, websites could easily use the width and the height of the rectangle object returned by the `getBoundingClientRect` method [2]. This method bypasses FireGloves fake dimension values and returns the text's dimension values even more precisely than the original methods did. Finally, FireGloves does not modifies the Flash API and therefore can websites still use Flash to discover the true nature of the browser, the real operating system, screen resolution, etc.

4.3.2 Private Browsing Modes

Almost all modern browsers support a private browsing mode. These modes aim to delete the tracks of a user's browsing activity and thus gives him more privacy on the web. They prevent the use of cookies, ETags and sometimes also the use of different plugins and add-ons for tracking purposes. Browsers also usually make sure that after leaving these modes, they wipe out the cache, the history, the cookies and other local storage spaces, in order to not leave any traces on the computer. However, all private browsing modes keep providing all the information needed for fingerprinting. Therefore, are users not protected by these modes against fingerprinting.

4.3.3 Do-Not-Track Header

The Do-Not-Track (DNT) HTTP header was introduced in 2009 by the researchers Christopher Soghoian, Sid Stamm, and Dan Kaminsky. It allows users to signal their tracking preferences to websites. Is the DNT header set to 0, then the user consents to being tracked. Is the DNT header set to 1, then the user does not want to be tracked. Is the DNT header missing in a request, then this means that the user has not expressed a preference. Since 2010 all major browsers support Do-Not-Track. The DNT header is currently being standardized by the W3C under the name "Tracking Preference Expression"[6]. However, DNT is not enforced, i.e. the server may choose to not honour the user's decision. This may work for honest website operators, but nevertheless, it is still up to them to decide if they stop tracking the user or ignore the request. For example, Acar et al. [2] have set their DNT header to 1 while their experiments with the FPDetective framework. They obtained the same results as with not having set the DNT header to 1, concluding that the DNT setting is ignored by fingerprinters. As a further example, Yahoo has recently decided to no longer enable the Do Not Track settings on their servers and therefore ignore the user's preference to not be tracked. Yahoo arguments that the DNT header is not effective, not easy to use and has not been adopted by the broader tech industry, whereas Yahoo was part of the first companies to implement Do-Not-Track[7].

[6]Tracking Preference Expression (DNT) – http://www.w3.org/2011/tracking-protection/drafts/tracking-dnt-last-call.html

[7]Yahoo's Default = A Personalized Experience – http://yahoopolicy.tumblr.com/post/84363620568/yahoos-default-a-personalized-experience

4.3.4 User-Agent Spoofing Tools

A browser's user-agent string is an important part of a fingerprint and thus it may seem reasonable to assume that if users modify their default values, they will increase their privacy [27]. Yen et al. [27] advice therefore the usage of user-agent spoofing tools. However, most user-agent spoofing tools have the following issues [20]:

- **Incomplete coverage of the navigator object.** User-agent spoofing tools which modify the `navigator.userAgent` property, mostly leave other revealing properties intact, such as `appName`, `appVersion`, `platform` and `vendor`.
- **Inconsistencies between related properties.** None of the user-agent spoofing tool attempts to modify the `screen` object. Thus users who are utilizing laptops or desktop computers and pretending to be mobile devices are reporting inconsistent screen resolutions.
- **Mismatch between user-agent values.** The user-agent of any given browser is accessible through the HTTP headers and through the JavaScript object `navigator.userAgent`. However, some of the user-agent spoofing tools might change the HTTP headers but leave out `navigator.userAgent` property.

As a result of these issues, is the presence of any user-agent spoofing tool a discriminatory feature. It makes them more visible and more distinguishable from other users, who are using their browsers without any modifications.

4.3.5 Tor Browser Bundle

The Tor Browser Bundle[8] is a popular service, which enables users to anonymously browse the web. It is based on a modified version of Mozilla Firefox and makes use of the Tor anonymity network. The Tor anonymity network relays communications over three routers located in different parts of the world. The communications are encrypted in layers, also denoted as onion routing, to prevent the Tor routers from linking the source and the destination of a data stream. The Tor Browser currently incorporates or plans to incorporate strong fingerprinting defenses against fingerprinting of: plugins, HTML5 canvas image extraction, WebGL, fonts, desktop resolution, CSS media queries and system colors, user-agent and HTTP headers, timezone and clock offset, JavaScript performance fingerprinting, non-uniform HTML5 API implementations and keystroke fingerprinting [23]. Fonts are operating-system and user dependent therefore they are an excellent candidate attribute for fingerprinting. The Tor Browser limits font-based fingerprinting similar to FireGloves, namely by limiting the number of fonts that a page can request and load. Since the Tor Browser uses the Tor anonymity network, which is sort of acting as a proxy between the user and the website, the real IP address of the user is therefore not revealed. In general the Tor Browser project can be seen as the "best" defence against browser fingerprinting at the moment, since it does the best effort to be anonymous. It changes the user-agent to a common browser version, it disables all plugins and it changes several properties to some default value. However, the Tor Browser users can still be identified as a single group. For instance, the available screen width and height as well as the total screen width and height are queryable through JavaScript.

[8]Tor Browser Bundle – https://www.torproject.org/projects/torbrowser.html.en

The Tor browser modifies the available screen size to the same value as the total screen size. This is quite unique since most browsers have a taskbar and therefore is the available screen height slightly less then the total screen height. Combined with the fact that most, if not all of the exit nodes from the Tor network are known, one could easily identify Tor users as a group.

Chapter 5

Methodology

In this chapter we discuss our approach regarding the separation of so called "web identities". As a first step we explain the notion of web identities and how the linking of web identities works. Afterwards, we mention some problems regarding the consistency of web identities. In addition, we discuss the approach of current browser fingerprinting countermeasures and then we emphasize on our own two possible approaches. Finally, we elaborate the setting of our two test cases, in order to validate our final approach and in order to allow the reproduction of our results for further research.

5.1 Web Identities

A web identity is an identity that is related to a user and is solely used on the web. Websites establish this identity in order to re-identify their users upon future interactions. This identity can be established through an IP-address, a cookie, a user-agent or simply through a user id when a user logs in to a website. In the context of browser fingerprinting, a website establishes a web identity through the information it gets about a user's browser characteristics: (e.g. user-agent, language, screen resolution, timezone, etc.). For our purposes, we focus solely on the HTTP and JavaScript inferable parts. Therefore, if a user changes his browser characteristics, then his web identity also changes. However, users do not frequently change their browser characteristics and therefore remains their web identity persistent. This is useful for advertising networks, since they can identify users without them being logged in.

5.2 Linking Web Identities

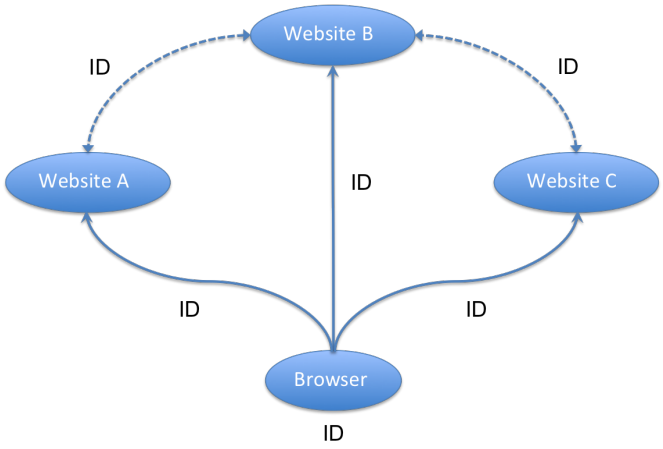


Figure 1: An example of linking web identities.

In order to track users across the web, websites match a newly received web identity ID_A with a previously established web identity ID_B. Websites usually do this by simply comparing if a received web identity is in their database of known web identities. However, a small change in the browser characteristics would change the web identity and therefore using a simple comparison does not work. Thus, we assume that most websites compare web identities from fingerprints through a matching score e.g. match(ID_A, ID_B) = 95%. As an example, consider Figure 1. The browser sends the same characteristics to website A, website B and website C. As a result, receive all three websites the same ID. Besides, we can also observe that the third-party website B receives the same ID from website A and from website C, which in turn means that the third-party website B knows that the user was on website A and on website C. This is also denoted as linking of web identities.

5.3 Maintaining Consistency between Web Identities and Browser Capabilities

Since web identities are partially bound to the browser characteristics, one obvious approach to privacy is tempting to simply change these characteristics. However, it is important to maintain consistency between reported information and actual browser capabilities in order to make oneself not conspicuous. For instance, a browser with JavaScript disabled which returns default values for screen resolution, browser plugins, system fonts and supercookies, indicates with the presence of these measurements that JavaScript is activated. More subtly, browsers with a Flash blocking extension such as Flashblock[1] installed will show Flash in the plugins list, but running Flash fails. Such behavioural quirks help to distinguish web identities even though neither measurement (browser plugins) explicitly detected the blocker. Similarly, many browsers with spoofed user-agent strings are distinguishable because other measurements are invalid for the reported

[1]Flashblock – https://addons.mozilla.org/en-US/firefox/addon/flashblock/

user-agent.

For example:

- Browsers sending iPhone user-agents but with the Flash player plugin installed (not supported on the iPhone)
- Browsers that identify themselves as Mozilla Firefox but support Internet Explorer userData supercookies.

5.4 Existing Countermeasures

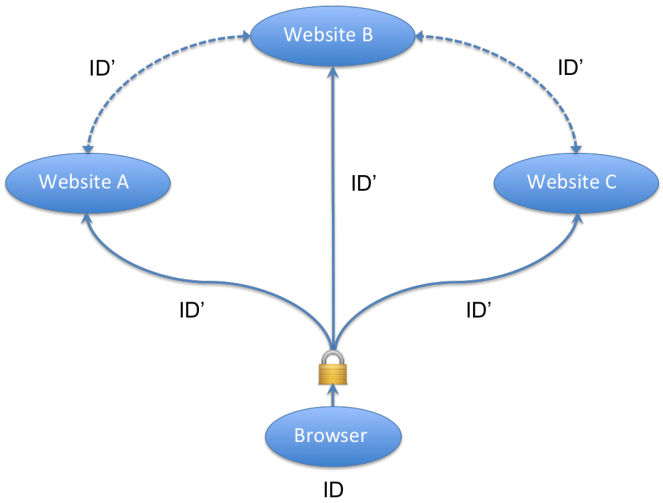


Figure 2: Web identities belonging to an anonymity set are still linkable.

Current browser fingerprinting countermeasures, such as FireGloves [3], aim to change the web identity of a user in such a way that he belongs to a large anonymity set. An anonymity set is in this case a set of users all having the same web identity and therefore not being distinguishable within this set. FireGloves achieves this by changing the user's browser characteristics to a very common one. Nevertheless, this approach has some downsides:

1. It requires a large number of users in order to work.
2. A user is required to frequently change his browser characteristics in order to keep being part of the majority of the users on the web.
3. A user is still unique through his IP-address.

As an example, consider Figure 2. FireGloves simply replaces the user's real web identity ID with ID'. Third-party website B receives ID' from both website A and website C, and is therefore able to link both IDs together. The user is identified as part of a set of users who visited website A and website C. However, each user can still be distinguished through his IP-address. As a result, we need a different approach in order to solve this issue with the linking of web identities.

5.5 Alternative Approaches to Web Privacy

In this part we elaborate in detail two possible approaches in order to prevent web identities to be linked. The first approach aims to hide where we are coming from and the second approach aims to completely separate web identities one from another.

5.5.1 Hide where the Request Originated

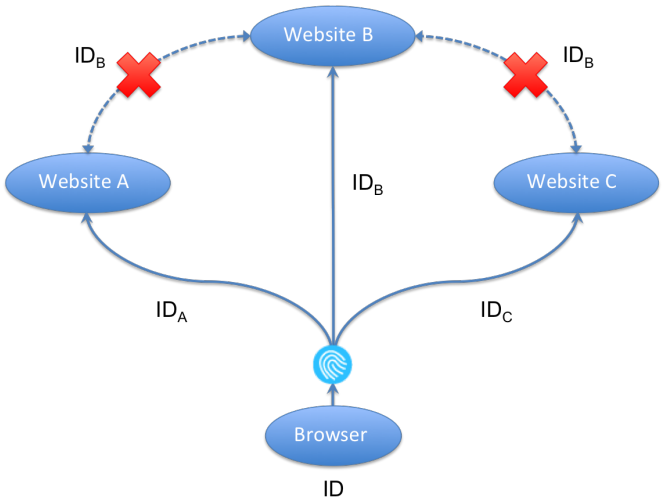


Figure 3: Hiding where the request originated.

A first possible approach is that we hide where each request originated. That is, regardless of this website acting as a third-party or not. For every website we first create a fake web identity (in Figure 3, ID_A, ID_B and ID_C). Next we prevent the connection (illustrated by a red cross) between website A and website B and between website C and website B. Afterwards we send from website A and website C to website B always ID_B. This means that for website B, every interaction looks as if it is directly communicating with the user and not via another website like A or C. This approach has two requirements in order to work properly:

1. We need to make sure that the fake web identities, which we created are not the same, i.e. $ID_A \neq ID_B$, and $ID_C \neq ID_B$, nor should they be trivially linkable; cf. Section 5.2.

2. We need to make sure that there is no revealing interaction between website A and website B (respectively website C and website B), such that website B does not know that it is actually interacting with the user through website A (respectively website C). Website A respectively website C and website B can interact in various ways, including:

   (a) HTML: Website A includes an url to website B referring itself. For example: `<img src="htttp://websiteB.org/refer/A" .../>`.
   Website B can parse this url and deduce website A as the origin of the request.

   (b) HTTP: The HTTP referrer header tells website B where a request originated.

(c) JavaScript, Flash, Silverlight and other client-side technologies: These can send direct requests to website B and tell website B the origin of the request through parameters (e.g. JavaScript - AJAX, Flash - Sockets).

(d) PHP, ASP, Java and other server-side technologies: These can send direct requests from website A to website B. For example:
`include 'http://websiteB.com/track.php?refer=A';`.

As the last item highlights, the interaction between first-party and third-party websites does not need to go via the client. Therefore, at this point it should be clear that this approach is not viable because we do not have any influence on the server-side. Finally it is simply impossible to always distinguish tracking content from necessary content, i.e. detect when a first-party website is revealing its true origin to a third-party website for tracking purposes.

5.5.2 Separate Web Identities

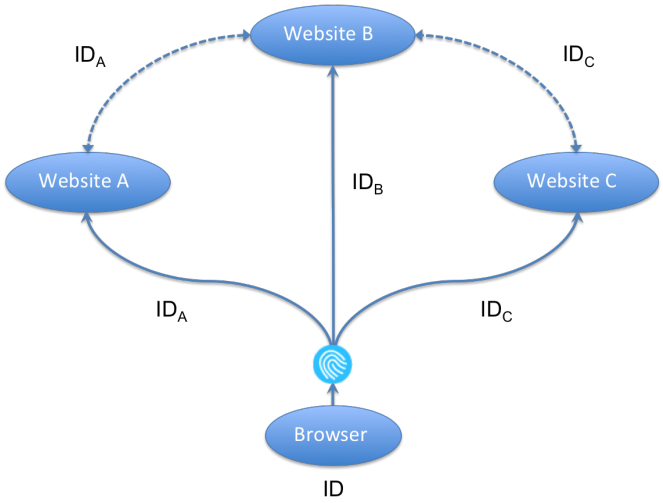


Figure 4: Separating web identities.

A second possible approach is to create a fake web identity for *each* website the user visits. Afterwards enforce that a third-party website receives the same fake web identity as the first-party website. For an example, consider Figure 4. When contacted directly, website A sees web identity ID_A, website B sees web identity ID_B and website C sees web identity ID_C. When website B is contacted through website A or website C, website B sees ID_A and ID_C, respectively. Website B sees therefore three individual and distinct web identities. As a result, website B cannot link ID_A with ID_B, nor can it link ID_C with ID_B. This approach has one requirement:

1. We need to make sure that the fake web identities, which we created are not the same, i.e. $ID_A \neq ID_B$, and $ID_C \neq ID_B$, nor should they be trivially linkable; cf. Section 5.2.

Although this is a strong requirement, seems this second approach to be promising, as we only need to address the first requirement as compared to the first approach.

5.5.3 Conclusion

We proposed two different approaches to web privacy. However, the first approach is not viable because we simply do not have any influence on the server-side and it is simply impossible to always distinguish tracking content from non-tracking content. Therefore we chose to incorporate the second approach in our Firefox extension, because it only requires us to make sure that we always generate unique fake web identities and enforce these properly.

5.6 Validation of the Implementation

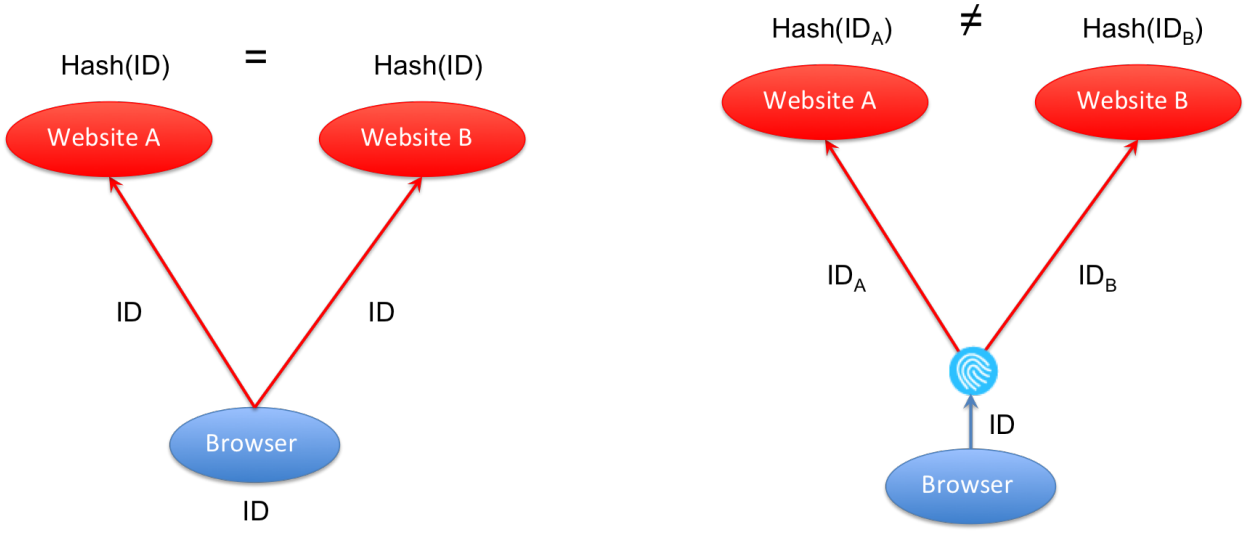
In order to validate our new approach of separating web identities, we need to test the following two cases:

1. The inclusion of a fingerprinting library on two distinct websites, simulating first-party fingerprinters.

2. The inclusion of a fingerprinting library on a website through a third-party website, simulating third-party fingerprinters.

5.6.1 Test Case 1: First-party Fingerprinters

The setting of our first test case is very simple but abundant. Website A and website B, both directly include a call to a fingerprinting library. Both websites display a hash value based on the web identity they detected. In the normal case, (cf. Figure 5a), we expect the two hashes to be the same. With the protection enabled, (cf. Figure 5b), we expect the two hashes to be distinct.



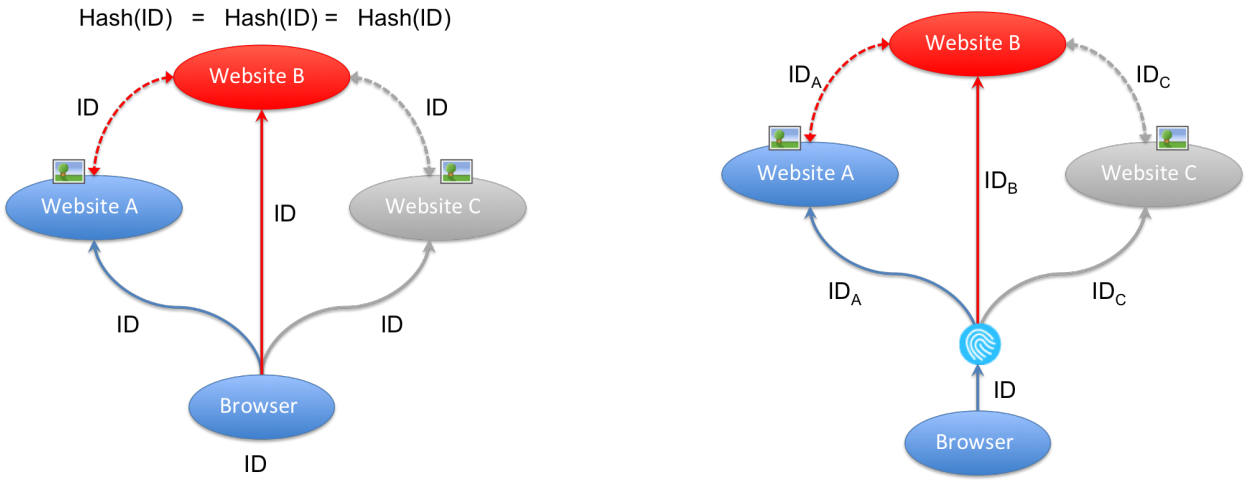
(a) Without the protection.

(b) With the protection enabled.

Figure 5: Test case 1: The inclusion of a fingerprinting library on two distinct websites.

5.6.2 Test Case 2: Third-party Fingerprinters

In our second test case only website B directly includes a call to a fingerprinting library. Website A and website C do not directly include any fingerprinting library, instead they include a third-party code from website B. Website C is grayed out because it is enough to only verify website A in order to show that third-party fingerprinting works. Website B displays a hash value from each web identity it detected. In the normal case, (cf. Figure 6a), we expect website B to log three equal hashes, one coming from website A, one coming from website B and one coming from website C. With the protection enabled, (cf. Figure 6b), we expect website B to log three distinct hashes, coming again from the three websites.



(a) Without the protection.

(b) With the protection enabled.

Figure 6: Test case 2: The inclusion of a fingerprinting library on a website through a third-party website.

Chapter 6

Development of *Fingerprint Privacy*

In this chapter we explain the development of our Firefox extension *Fingerprint Privacy*. Firefox extensions or sometimes also denoted as "add-ons", are installable enhancements for Firefox, which modify the behaviour of existing features or add entirely new features. Firefox extensions are developed through XML, JavaScript and XUL. XUL stands for XML User Interface Language and is solely developed by Mozilla to create interfaces in Firefox and Thunderbird.

We describe first the structure of a Firefox extension in general and how the packaging and installing of a Firefox extension is done. Afterwards we explain the interception and modification of HTTP requests respectively responses. Furthermore we explain the injection of JavaScript code in order to override properties of the JavaScript API. Moreover, we outline the implementation of web identities in our extension and we outline the implementation of our random fingerprint generator. In addition, we emphasize on the detection of fingerprinting activities and the options we provide to the user upon a detection. Furthermore, we highlight how we handle requests to social plugins and browser plugins. On top of that, we discuss the implementation of our user interface. Finally, we explain the validation of our extension and list the associated limitations and dependencies.

6.1 Structure of Firefox Extensions

Every Firefox extension is build up with a defined hierarchical folder structure as illustrated in Figure 7. In this section we solely describe the purpose of the individual folders and files. We recommend you therefore to skip this section if you are not necessarily interested in the structure of Firefox extensions.

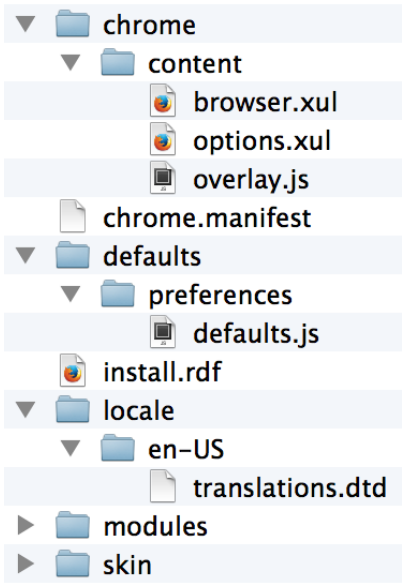


Figure 7: An example of the hierarchical folder structure of a Firefox extension.

6.1.1 Chrome Folder

The chrome of Firefox consists of everything around the content window, e.g. web browser toolbar, menus, statusbar etc. The chrome folder is composed of a folder *content*, which is responsible for the user interface and the main business logic of the extension. The following three files must exist within the *chrome/content* folder:

Browser.xul
: This file overrides some of the default look of the web browser, e.g. add a button to the toolbar, an item to the Tools menu or add a status-bar icon. This file also defines the component structure of the popup menu of the toolbar button.

Options.xul
: This file defines the structure and components of the options or preferences dialog of the extension.

Overlay.js
: This file handles the core business logic of the extension and allows to script for any element defined inside the *browser.xul* file.

6.1.2 Chrome.manifest

The *chrome.mainfest* file is in conjunction with *install.rdf* the key to how the extension will be added to Firefox. Furthermore describes the *chrome.mainfest* file the paths to the individual files which are necessary for the extension.

6.1.3 Defaults

The *defaults* folder is responsible for the default content of the extension. It contains a folder *preferences*, which itself contains a file *defaults.js*. This file *defaults.js* is used to store the preferences of the extension.

6.1.4 Install.rdf

The *install.rdf* file contains all the meta information about the extension, which versions of Firefox it supports and other assorted information.

6.1.5 Locale

The *locale* folder is used for localization. In this case there is only one child folder, namely one for American English content. However, this can be easily extended for other languages. The *en-US* folder has a file *translations.dtd*, which contains translations for the labels used in XUL files.

6.1.6 Modules

The *modules* folder contains JavaScript modules which are required by the business logic of the extension.

6.1.7 Skin

The *skin* folder contains the images and the CSS files which are used to describe the appearance of all XUL components and therefore altering the look and feel of the extension.

6.2 Packaging and Installing

Firefox extensions are delivered as XPI files. XPI files are basically just ZIP files with another extension. Therefore, all we need to do is ZIP all files together and give it an XPI extension. Note that we do not ZIP the containing folder of the extension, just its contents (e.g. *chrome* folder, *chrome.manifest*, *install.rdf*, etc).

The installation is fairly straightforward. Once we created the XPI file, we just need to drag and drop the file into Firefox and it will automatically install.

6.2.1 Packaging with Windows

We select all the contents of the extension folder, do a right-click and choose: *Send To* → *Compressed (Zipped) Folder*. Afterwards we simply rename the resulting ZIP file to .xpi instead of .zip.

6.2.2 Packaging with Mac

We open the Terminal, navigate to the extension folder with the *cd* command and then we type in the following command: `zip -r ../fingerprintprivacy.xpi * -x *.DS_Store*`.

6.2.3 Packaging with Linux

We open a console, navigate to the extension folder with the *cd* command and we type in the following command: `zip -r ../fingerprintprivacy.xpi *`.

6.3 Embedding a Proxy

Our approach of separating web identities requires us to have full control over the communication between the browser and the website. The communication between a browser and a website is entirely done through HTTP. The browser sends an HTTP request to the website and the website answers with an HTTP response. Therefore, we embedded a proxy in our Firefox extension, which allows us to intercept and modify these HTTP responses and HTTP requests. As an example, consider Figure 8. The proxy of our extension receives the original request from the browser to website A, modifies this request, and forwards the request to website A. Afterwards, receives our proxy the original response from website A, modifies this response and forwards it to the browser.

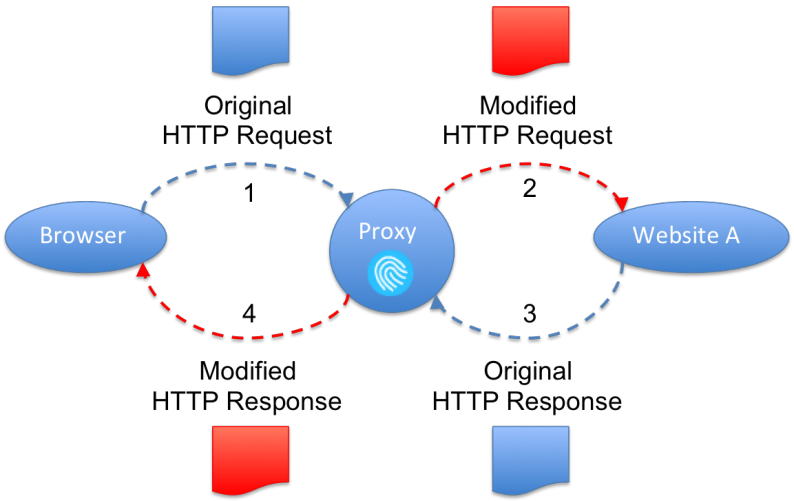


Figure 8: A representation of the workflow of our embedded proxy.

6.3.1 Intercepting and Modifying HTTP requests

There are several ways to detect and intercept loading of web pages and their contentIn order to modify any HTTP request header, we implemented an HTTP observer object. This object listens to the "http-on-modify-request" event and notifies our extension when a HTTP request is ready to be send and is available for the modification of its HTTP headers. Once our extension is notified, we do the following steps:

1. Determine the requesting and the referring domain.

2. Look up if there is already an existing web identity for the referring domain.
   (a) If there is a web identity, we simply get its fingerprint.
   (b) If there is no web identity, we create a new web identity and generate a new fingerprint. Note that our main goal is to keep web identities separate. In order to achieve this we must return a unique fingerprint for each domain we visit. Therefore, when generating a new fingerprint, we ensure that the fingerprint is different from the fingerprints of all the other domains.
3. For the `user-agent`, the `accept-language` and the `accept-encoding` header:
   (a) In case spoofing, we replace the original values with the generated values of our fingerprint.
   (b) In case blocking, we remove the headers from the HTTP request.
   (c) In case allowing, we just keep the original values.
4. Remove the following ETag headers in order to prevent fingerprinting:
   - If-Match
   - If-None-Match
   - If-Range

   Note that ETag headers allow web caches to be more efficient and therefore removing these headers might have an impact on the bandwidth and websites might load more slowly. However, ETags can also be used to fingerprint users. Furthermore, we chose for a blacklisting approach, because websites can use their own headers (headers starting with an "X-") and therefore make it simply impossible to go for a whitelisting approach.
5. Check if the requesting domain is different from the referring domain. In case of both being different:
   (a) Classify the requested domain as a third-party and add it to the list of third-parties for this particular web identity.
   (b) Remove the HTTP *referer* header from the HTTP request.
   (c) If the requested domain is part of a known social network (cf. Section 6.7), block and reject the HTTP request.

6.3.2 Intercepting and Modifying HTTP responses

In order to modify HTTP responses, we implemented again an observer object. This object however, listens to the "http-on-examine-response" event and the "http-on-examine-cached-response" event and notifies our extension when a response is ready to be downloaded from the server. Once notified, we need to filter out every response which is not an HTML file. Afterwards, before the file is parsed by the JavaScript engine, we inject JavaScript code that overrides properties of the `navigator`, `screen` and `Date` object (see Appendix B for a more detailed list). This

JavaScript code is injected on top of the HTML file, just right after the `<head>` tag, in order to be parsed first. A part of the injected code consists of sending a notification to our extension when a property is called (cf. Section 6.6). Another part of our injected code blocks JavaScript access to browser-identifying properties or returns fake values for these. In the end our injected JavaScript code removes itself from the DOM parse tree, as similar to FireGloves [3], in order to prevent other websites to detect our injected code through JavaScript.

6.4 Implementation of Web Identities

Web identities are implemented as JavaScript objects. A web identity is always associated to a particular domain (e.g. `example.com`) and consists of a list of attributes representing a fingerprint, a list of third-parties, a list of allowed respectively blocked social plugins and a list of allowed respectively blocked browser plugins. Once a web identity is generated, it is saved and always reused for that particular domain. Each web identity is added to an array which contains all of the web identities. When the user shuts down the browser, the web identities are stored as an JSON array in the *defaults.pref* file inside the *defaults/preferences* folder. When the user starts up the browser, the web identities are loaded from the JSON array and parsed back to JavaScript objects. Web identities can be removed and regenerated. Furthermore, can individual attributes of the attribute list which represents the fingerprint, be edited, removed and regenerated.

6.5 Random Fingerprint Generator

The random fingerprint generator used by our extension is based on Jeffrey Mealo random user-agent generator[1]. This random user-agent generator was extended in order to generate random fingerprints. A generated fingerprint consists of:

- a `user-agent` string
- a `navigator` object containing the following properties:
  - App Code Name
  - App Name
  - App Version
  - Language
  - Platform
  - OS CPU
  - Product
  - Vendor
- a `screen` object containing the following properties:
  - Screen Height

[1] random-ua.js - A random User-Agent Generator v0.0.4 – https://github.com/jmealo/random-ua.js

  - Screen Width
  - Color Depth
  - Available Height
  - Available Width
  - Pixel Depth

- a `Date` object containing a timezone offset
- an accept object containing an `accept-encoding` string

The user-agent string is generated by first randomly choosing a particular browser from a defined list of browsers:

- Mozilla Firefox
- Google Chrome
- Apple Safari
- Opera

Afterwards an operating system is randomly chosen from a defined list of operating systems:

- Microsoft Windows
- Mac OS X
- Linux

As a next step, a random version number is generated for the browser and the operating system. The random version numbers are bound to a range belonging to a particular browser or operating system. For example, if we randomly choose as browser Safari, then we will generate a random Apple Web Kit version number as follows: [531...536].[1...50].[1...7] and a random Safari version number as follows: [4...7].[0...1].[1...5]. If we randomly choose as operating system Windows, then we will generate a random Windows NT version number as follows: [5...6].[0..2]. Finally, the user-agent string is constructed based on the chosen browser and operating system, for example:

```
Mozilla/5.0 (Windows; U; Windows NT 6.1) AppleWebKit/535.14.5 (KHTML, like
Gecko) Version/6.0.5 Safari/535.14.5
```

Moreover, all the `user-agent` related JavaScript `navigator` object properties are set according to the generated user-agent string. The language property of the `navigator` object, is chosen randomly from a list of 96 languages.

The screen height and the respective screen width are chosen at random from a defined list of screen resolutions:

- 1366x768
- 1024x768
- 1280x800
- 1920x1080
- 1280x1024
- 1440x900
- 1600x900
- 1680x1050
- 1360x768

These screen resolutions originate from the 10 most used desktop screen resolutions between 04/2013 and 04/2014[2]. The color depth and the pixel depth are equivalent and are randomly chosen from three possible values: 16-bit, 24-bit and 32-bit. The available screen width is set to the same value as the screen width. The available screen height is set to a smaller height than the actual screen height, by taking the screen height and subtracting 74 pixel from it. This should simulate the toolbar of the browser. Note that changing the screen resolution can result in undesired usability effects.

The timezone offset is generated by randomly choosing a timezone offset from a list of all 40 possible existing timezone offsets (e.g. -720, -660, -600, -570, ..., 720, 765, 780, 840).

The HTTP `accept-encoding` string is generated according to the browser we chose. If the browser is Opera we return "gzip,deflate,lzma,sdch", if the browser is Chrome we return "gzip,deflate,sdch" and for the other browsers we always return "gzip, deflate".

The random generator makes use of the JavaScript `Math.random()` method, which is seeded from the current time, as in Java. In Firefox is the `Math.random()` method implemented as a linear feedback shift generator similar to the one of Java[3].

6.6 Detecting Fingerprinting Activities

In order to receive notifications from the DOM tree, we implemented an event listener in our extension which listens to an event called "DetectionEvent". When a website calls one of the properties from the `navigator`, `screen` or `Date` object (see Appendix B for a more detailed list), our injected code is called instead and sends a notification to our extension about the call through the "DetectionEvent". The extension then notifies the user by showing a notification

[2]Top 10 screen resolutions 04/2013 - 04/2014 – http://gs.statcounter.com/#desktop-resolution-ww-monthly-201304-201404

[3]Firefox Math.random() – http://mxr.mozilla.org/mozilla-central/source/js/src/jsmath.cpp#728

banner (cf. Figure 9). The notification banner lists all the attributes which got called by the website and proposes the following two options to the user:

Allow
A popup window is displayed where the user can individually allow certain attributes to be read.

Keep blocking
All attributes are blocked and the banner closes.

The extension remembers the option that the user took for a particular website. Later on the user is only notified if a website is trying to request additional attributes.

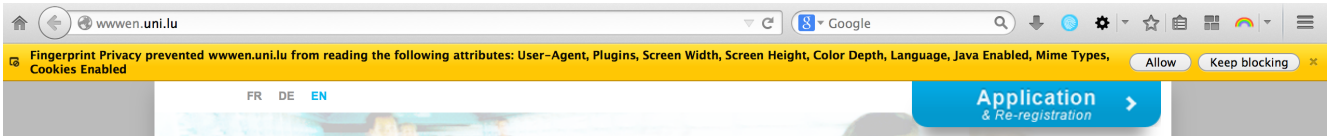


Figure 9: Notification banner on wwwen.uni.lu showing the list of detected attributes and the two options.

6.7 Handling Requests to Social Plugins

Figure 10: Examples of social plugins.

A social plugin, also sometimes denoted as social widget, is a piece of JavaScript code or HTML code embedded on a website in order to provide a particular service from a social network. Examples of such services are the Facebook Like button, the Twitter Tweet Button, the Google +1 button or the Pinterest Pin it button (cf. Figure 10). These social plugins are quite common these days, a large amount of websites include them in their source code. However, as discussed in Section 2.2.4, social networks may use these social plugins as web bugs in order to track which websites a user has visited.

Our extension simply blocks these social plugins by rejecting the HTTP requests for particular known social scripts from social networking websites. We are blocking requests to the following social networks and particular social scripts:

- Facebook [9]
  - `connect.facebook.net/en_US/all.js`
  - `facebook.com/plugins/`

- Twitter [13]
  - `platform.twitter.com/widgets.js`
  - `platform.twitter.com/widgets/`
- Google+ [10]
  - `apis.google.com/js/platform.js`
  - `apis.google.com/js/plusone.js`
- LinkedIn [5]
  - `platform.linkedin.com/in.js`
- Tumblr [12]
  - `platform.tumblr.com/v1/`
- Pinterest [11]
  - `assets.pinterest.com/js/pinit.js`

As an example for Facebook, we are blocking the Facebook Like button by checking if the requested domain from the HTTP request contains "facebook.com/plugins/" or contains "connect.facebook.net/enUS/all.js". If it does, we simply reject the request. Furthermore, we picked these social networks because these provide a dedicated API for developers and because these are widely used on the web.

We compared our approach with the one of Ghostery[4] and Disconnect[5]. Ghostery is a free browser extension owned by an advertising company, which enables its users to detect and control web bugs. Ghostery does not fully replace the social plugin buttons on a website. Ghostery puts its own button, which once clicked, simply allows the social networking website to load its own button. This is basically the same as our implementation. Disconnect is a free open source browser extension created by former Google engineers, which enables users to block third-party content and block social plugins from Facebook, Twitter and Google. Disconnect uses the same approach as we do, it simply blocks the requests to known social plugins and displays therefore no social plugins on a website.

However, a better approach would be completely replacing any social button with our own button and not let the social networking website load its own code. If the user then clicks our own button, we directly forward the request to the social networking website and hide where the request is originally coming from. Obviously should the user have the same experience as if he would have clicked on the original button of the social networking site.

[4]Ghostery – https://www.ghostery.com/en/
[5]Disconnect – https://github.com/disconnectme/disconnect

6.8 Handling Requests to Browser Plugins

The list of installed browser plugins can provide a lot of information since the list and the order of installed plugins is different from browser to browser. Especially the order of installed plugins is a worthwhile characteristic for fingerprinting. When a plugin is installed or updated it is added to the top of the list of plugins, therefore two users with the same installed plugins might still have differently ordered lists of plugins. In JavaScript can the list of installed browser plugins be retrieved through the `navigator.plugins` property and the `navigator.mimeTypes` property. Both properties return an array containing information about the installed browser plugins including their mime types.

Our extension solves these issues with the order and the list of the installed plugins, by blocking browser plugins individually and by keeping the order consistent. Only the QuickTime Player[6], Flash Player[7], VLC Player[8] can be blocked individually, because these are the frequently used ones on websites for multimedia content. However, all browser plugins are blocked by default and therefore the list of plugins is empty. If a plugin is allowed for a website, then a new fake plugin list is created for this website and the allowed plugin is added to the list and returned when requested.

Note that we actually do not completely block these browser plugins. We solely prevent them to be read through the JavaScript API. In other words, the browser plugins will still work also if we block them with our extension. However, it turns out that many websites check first through JavaScript if a particular plugin is present or not. If the plugin is missing then they do not run the plugin and ask the user to install the plugin, while the plugin is actually installed.

6.9 User Interface

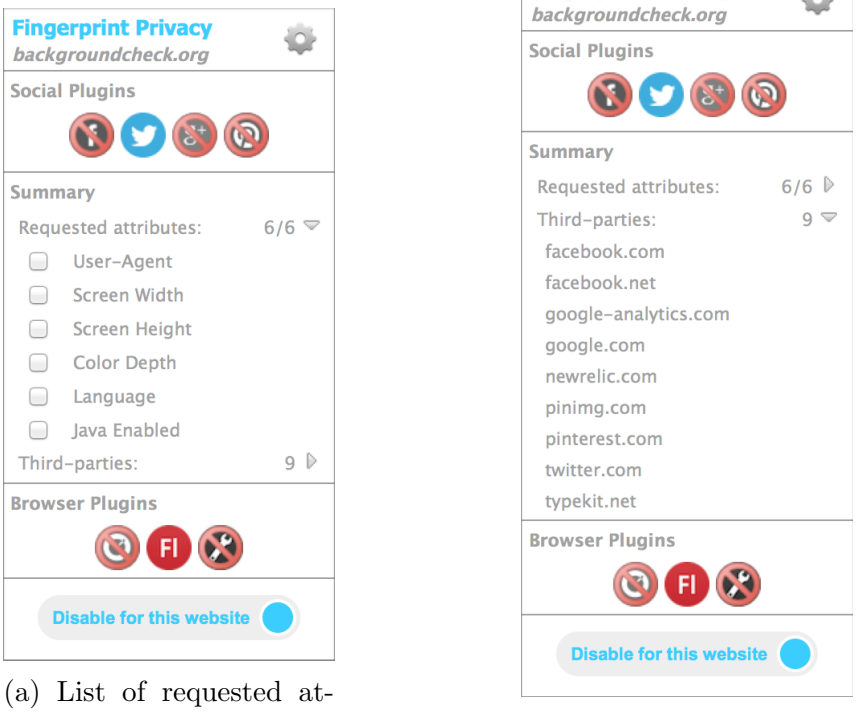
In this section we discuss the user interface of our Firefox extension *Fingerprint Privacy*. We highlight the different functionality available through the toolbar button popup menu and the preferences window.

6.9.1 Toolbar Button Popup Menu

The popup menu of the toolbar button is divided into five different parts. The first part of the popup menu displays the domain name of the current open website and provides the user a direct way to the preferences window by clicking on the gear icon on the top right. The second part of the popup menu is dedicated to social plugins. The extension detects which of our known social plugins the current website is using and provides the possibility to the user to individually block respectively allow them for the current domain. A social plugin is blocked if it is greyed out and struck through. The third part of the popup menu displays a short summary regarding the number of attributes the current website requested and the third-parties that the extension detected. The user can allow respectively block individually the requested attributes by simply expanding the

[6]QuickTime Player – https://www.apple.com/quicktime/what-is/
[7]Flash Player – https://www.adobe.com/software/flash/about/
[8]VLC Player – http://www.videolan.org/vlc/index.html

list and clicking on the according checkbox (cf. Figure 11a). Furthermore, by expanding the list of third-parties the user can view an alphabetic sorted list with the domain names of the third parties (cf. Figure 11b). The fourth part of the popup menu is dedicated to browser plugins. The user has the possibility to individually block respectively allow multimedia content plugins and block respectively allow non multimedia content plugins for the current domain. The fifth and last part of the popup menu allows the user to disable respectively enable the extension for the current website.



(a) List of requested attributes.

(b) List of third-parties.

Figure 11: User interface for the toolbar button popup menu.

6.9.2 Preferences Menu

The preferences menu allows the user to disable respectively enable the notifications of requested attributes from websites. Moreover, allows the preferences menu to open the web identities management menu in order to manage individual web identities and their corresponding attributes. Furthermore, is the preference menu divided into two groups of settings, namely HTTP settings and JavaScript settings. The user can in the HTTP settings block respectively allow third-party cookies, remove respectively keep the HTTP Referer header, send respectively do not send the Do Not Track header and finally remove respectively keep the HTTP ETag headers. The user can in the JavaScript settings check if he wants to automatically block all known social plugins and if he wants to automatically block all browser plugins. Finally, has the user the ability to reset all settings back to their default values.

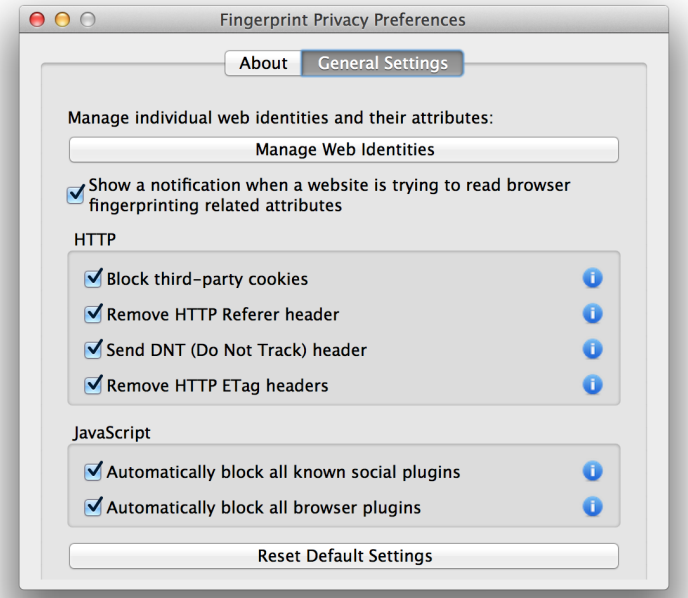


Figure 12: User interface for the preferences menu.

6.9.3 Web Identities Management Menu

The web identities management menu lists all the generated web identities. Each web identity belongs to a particular domain. The user can expand a web identity in order to see its full list of attributes. Each attribute has a name, a value and an action. If the user does a right-click on a web identity, then a dedicated context menu appears and allows the user to view the respective third-parties, delete the web identity and regenerate the web identity. If the user does a right-click on an attribute, then a different dedicated context menu appears and allows the user to open the edit menu for the attribute, delete the attribute and regenerate the attribute.

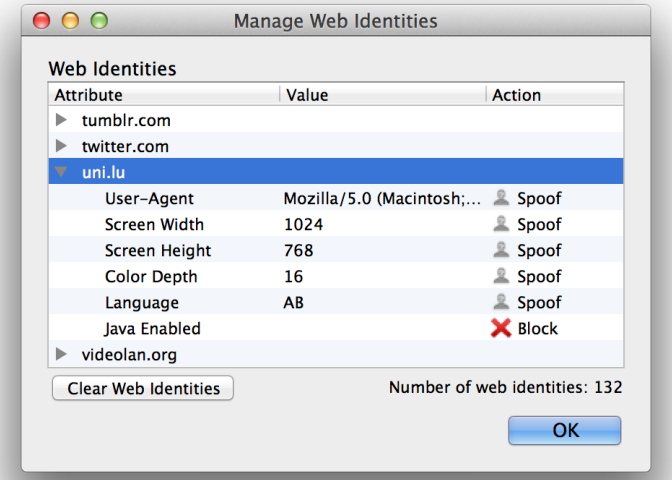


Figure 13: User interface for the web identities management menu.

6.9.4 Edit Attribute Menu

The edit attribute menu allows the user to edit the action and value field of a selected attribute. An attribute may have three possible actions: *allow* – the website is allowed to read the attribute, *spoof* – the website receives the fake value of the attribute and *block* – the website has no access to the attribute. Whereby some attributes only have two possible actions, *allow* and *block*, because our fingerprint generator does not generate a value for them and therefore making the action *spoof* useless. The value of an attribute can only be modified when the action *spoof* is selected.

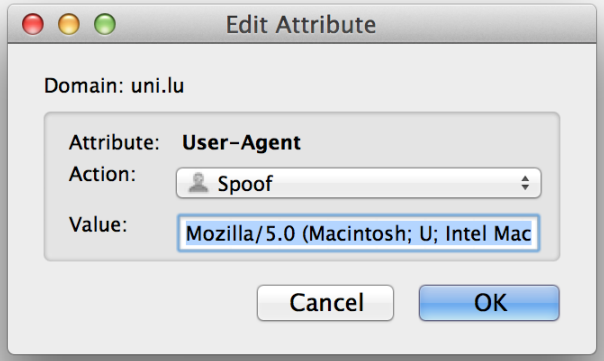


Figure 14: User interface for the edit attribute menu.

6.10 Validation of the Extension

As previously described in Section 5.6, we set up two test cases with two different websites, in order to test our new approach of separating web identities. In this section we will describe the implementation of both test cases. As fingerprinting algorithm we use the fingerprinting library "fingerprintJS"[9] from Valentin Vasilyev. FingerprintJS is a fast open source browser fingerprinting library written in pure JavaScript with no dependencies. It collects browser information through JavaScript and hashes all gathered information into a 32-bit integer. It uses Murmur hashing, which is a non-cryptographic hashing function created by Austin Appleby in 2008.

6.10.1 Test Case 1: First-party Fingerprinters

In our first test case, we want to test first-party fingerprinters. Therefore, we embed on both websites directly in their source code the fingerprinting library. Both websites create a fingerprint and display the 32-bit hash value based on the web identity they detect. Before testing our extension, we had to make sure that the fingerprinting library works. We did this by comparing the hash values displayed on both websites. If both hash values are the same, then it means that the fingerprinting library is working. Afterwards we enabled our extension and reloaded both websites. We compared both hash values and observed that they were this time different from each other. This proves that our extension successfully prevents fingerprinting for our first test case.

6.10.2 Test Case 2: Third-party Fingerprinters

In our second test case, we want to test third-party fingerprinters. Website A includes therefore an iframe which makes a call to a PHP file located on the server of website B. The PHP file on website B generates an HTML page including the desired picture for website A and it also embeds in the HTML code the fingerprinting library. When the picture is loaded on website A, a hash of our fingerprint will be created and sent together with the IP-address and host domain to website B through AJAX. Website B receives this information and logs it into a file. When visiting website B, it displays the 10 most recent entries of the file and it also displays the hashed value of our current fingerprint. We did the test first with our extension disabled. If the hash on website B is the same as the hash received from website A, then it means that the third-party fingerprinting is working. Afterwards, we enabled our extension and reloaded first website A and then website B. We compared if our hash was corresponding to the hash received from website A and observed that both were distinct. This proves that our extension successfully prevents fingerprinting for out first test case.

6.11 Limitations of the Extension

Firstly, we decided to solely focus on HTTP and JavaScript fingerprinting techniques due to the given time constraints that we had. The extension can thereby not guarantee with absolute

[9] fingerprintJS v0.5.3 – https://github.com/Valve/fingerprintjs

certainty that it is still effective against commercial fingerprinting methods, since these make excessively use of Flash. The Flash API provides the same properties as the JavaScript API, whereas the Flash API returns for some properties more detailed information than the JavaScript API. In addition is the fingerprintable surface larger at Flash. Therefore will websites that make use of Flash still be able to fingerprint their users.

Secondly, we do not provide a complete protection against social plugins. We only handle social plugins from six different social networks, namely: Facebook, Twitter, Google, LinedIn, Tumblr and Pinterest. Other social plugins can therefore still track users across the web.

Thirdly, we solely block the list of install browser plugins from being read through the JavaScript API. This means that browser plugins can still be executed through their own API calls.

Fourthly, we tested our extension on different platforms such as Windows, Linux and MAC OS X. We realised however that our extension only works on MAC OS X, probably because we developed our extension on MAC OS X. Nevertheless, we were expecting our extension to work across different platforms, since this is one of the purposes of an Firefox extension. A Firefox extension should run on every platform running Firefox.

Finally, if a fingerprinting website finds a way to detect our extension then this website can use this additional information as a distinctive feature for identification.

6.12 Dependencies

Our Firefox extension has solely one dependency, namely jQuery[10]. We make use of jQuery 2.1.0 and the library is directly bundled with the extension and therefore does not need to be updated or added manually.

[10]jQuery – https://github.com/jquery/jquery

Chapter 7

Discussion and Future Work

Fingerprint Privacy successfully prevents, under the tested circumstances, browser fingerprinting through libraries using HTTP and JavaScript. The extension uses our new approach of separating web identities. It generates for each visited website a unique random web identity. This prevents websites from linking web identities since the ones we provide are unique and therefore completely different from one another. The new approach that we proposed is generic and can be further extended to prevent fingerprinting through any client-side scripting technology. In addition, we block third-party cookies through the built-in option of Firefox. Finally, we block social plugins and prevent installed browser plugins from being detected through JavaScript.

7.1 Improving Existing Functionality

Social plugins from Facebook, Twitter, Google, LinkedIn, Tumblr and Pinterest are successfully blocked. However, this finite list could be extended to include more social networks and as described in Section 6.7, a better approach would be replacing the original buttons through our own buttons.

Browser plugins are not completely blocked but only removed from the list of plugins which is retrievable through the `navigator.plugins` object and the `navigator.mimeTypes` object. We could improve this by also blocking the execution of browser plugins.

The list of JavaScript attributes that our extension currently detects could be further extended. For example, we observed that some commercial fingerprinting libraries use attributes such as: math constants, date, time and the AJAX implementation of the browser to further fingerprint users.

The extension was mainly developed and tested on MAC OS X. We also tested our extension across different platforms such as Windows and Linux. However our extension did not work on the latter platforms. This issue could be solved through further development and testing on these platforms.

7.2 Adding New Functionality

We had to limit the scope of our extension due to time constraints. Therefore we only focused on HTTP and JavaScript fingerprinting. However, commercial fingerprinting libraries do not solely use HTTP and JavaScript but also make use of Flash to fingerprint their users. For that reason we aim to extend our extension in order to detect websites requesting attributes through Flash and to find a way to override the original Flash attributes with our own attributes.

Firefox claims to manage Flash cookies the same way as normal HTTP cookies. However, this is only the case for the deletion of Flash cookies. In order to block third-party Flash cookies, we must use a separate tool such as Better Privacy[1] or the Settings Manager from the Flash Player[2]. We are aware that this creates a potential risk and that our extension should be extended in order to also block Flash cookies.

Current fingerprinting countermeasures such as FireGloves [3] focus mainly on font detection. Previous research has shown [4, 7, 20] that fonts leak a lot of identifiable information. We did not investigate further into font detection because it was out of our scope and FireGloves already tries to combat this. However font detection can be done through Flash and JavaScript, whereas FireGloves only focuses on JavaScript font detection. Therefore a possible improvement for our extension could be including font detection detecting code, e.g. based on FireGloves, in order to prevent font detection through JavaScript and extend this code to also prevent font detection through Flash.

Fingerprint Privacy detects third-party requests on websites but does not provide any further information. We could try to classify such requests through online databases such as Google's Safe Browsing[3] or Trend Micro's Site Safety Center[4]. These online services return if a website is safe, harmful, suspicious, or untested. However, the disadvantage is that these online databases would receive all of our requests and we do not want to reveal every domain we visit because this would induce a privacy risk. We are not sure to what extend these online services might use our information. Therefore a better approach would be for the extension to compile such a list from online sources and perform the check offline.

An additional improvement could be the use of an empty list of web identities per private window, instead of using the existing ones, when the user is using the private browsing mode of Firefox. Private browsing allows users to browse the web without saving any information about which websites they visited and our extension should comport with this. Moreover could the extension's user interface be improved in order to achieve a better user experience.

Finally, we would like to test our extension against commercial fingerprinting re-identification services.

[1] Better Privacy – https://addons.mozilla.org/en-US/firefox/addon/betterprivacy/

[2] Disable third-party local shared objects – http://helpx.adobe.com/flash-player/kb/disable-third-party-local-shared.html

[3] Google Safe Browsing API – https://developers.google.com/safe-browsing/

[4] Trend Micro Site Safety Center – http://global.sitesafety.trendmicro.com/

Chapter 8

Conclusion

We researched web privacy with a focus on browser fingerprinting. We discussed in detail the context of web tracking, identifying the principle reasons for web tracking and discussing the evolution of user tracking. Then we outlined the definition and characteristics of browser fingerprinting, where we described the differences between passive and active fingerprinting methods. In addition, we elaborated the robustness of browser fingerprints and discussed fingerprinting from the perspective of the OSI model.

Afterwards we discussed Panopticlick's [7] fingerprinting algorithm and considered commercial fingerprinting libraries discovered by Nikiforakis et al. [20] We analysed and evaluated current fingerprinting countermeasures, such as Tor [23], FireGloves [3, 4] by Boda et al. and user-agent spoofing tools and highlighted their current limitations. Such countermeasures are easily detectable and therefore easily breakable. Often, countermeasures aim to hide the web identity of a user by modifying a user's browser characteristics in such a way that these are identical to the characteristics of a large set of users. We decided to do the opposite: instead of providing a *generic web identity for every website*, we provide a *unique web identity for each website*.

Finally, we developed a Firefox extension called *Fingerprint Privacy*, which applies our new approach to browser fingerprinting. We validated our extension with the help of two test cases. The first test case simulates first-party fingerprinting, the second test case simulates third-party fingerprinting. In both test cases we used an open source fingerprinting library written in JavaScript. Our extension successfully prevented this library from linking two fingerprints in both test cases.

Fingerprint Privacy currently combats re-identification only through HTTP and JavaScript. Commercial fingerprinting libraries make excessive use of Flash to fingerprint users. Therefore the next step is to adapt our Firefox extension to also account for Flash.

Bibliography

[1] 41st Parameter. Complex Device Recognition. http://www.the41.com/solutions/complex-device-recognition. Retrieved April 6, 2014.

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

[3] Károly Boda, Ádám Máté Földes, and Gábor György Gulyás. FireGloves – Cross-browser fingerprinting test 2.0. http://fingerprint.pet-portal.eu/?menu=6. Retrieved May 10, 2014.

[4] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. User Tracking on the Web via Cross-Browser Fingerprinting. In *Proceedings of the 16th Nordic Conference on Information Security Technology for Applications*, NordSec'11, pages 31–46, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] LinkedIn Corporation. LinkedIn Share button documentation. https://developer.linkedin.com/share-plugin. Retrieved May 30, 2014.

[6] Peter Eckersley. A Primer on Information Theory and Privacy. https://www.eff.org/deeplinks/2010/01/primer-information-theory-and-privacy, January 2010. Retrieved April 6, 2014.

[7] Peter Eckersley. How Unique Is Your Web Browser? In *Privacy Enhancing Technologies*. Springer, 2010.

[8] Jeremiah Grossman. CSS History Hack – I know where you've been. http://jeremiahgrossman.blogspot.com/2006/08/i-know-where-youve-been.html, August 2006. Retrieved April 6, 2014.

[9] Facebook Inc. Facebook Like button documentation. https://developers.facebook.com/docs/plugins/like-button/. Retrieved May 9, 2014.

[10] Google Inc. Google +1 button documentation. https://developers.google.com/+/web/+1button/. Retrieved May 9, 2014.

[11] Pinterest Inc. Pinterest Pin It button documentation. https://developers.pinterest.com/pin_it/. Retrieved May 30, 2014.

[12] Tumblr Inc. Tumblr Share button documentation. `http://www.tumblr.com/buttons`. Retrieved May 30, 2014.

[13] Twitter Inc. Twitter Tweet button documentation. `https://dev.twitter.com/docs/tweet-button/`. Retrieved May 9, 2014.

[14] Samy Kamkar. Evercookie - Virtually Irrevocable Persistent Cookies. `http://www.samy.pl/evercookie/`, September 2010. Retrieved April 4, 2014.

[15] Michael L. Kent, Bryan J. Carr, Rebekah A. Husted, and Rebeca A. Pop. Learning web analytics: A tool for strategic communication. *Public Relations Review*, 37(5):536–543, December 2011.

[16] Tadayoshi Kohno, Andre Broido, and K.C. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, May 2005.

[17] Keaton Mowery, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. Fingerprinting Information in JavaScript Implementations. In Helen Wang, editor, *Proceedings of W2SP 2011*. IEEE Computer Society, May 2011.

[18] Keaton Mowery and Hovav Shacham. Pixel Perfect: Fingerprinting Canvas in HTML5. In Matt Fredrikson, editor, *Proceedings of W2SP 2012*. IEEE Computer Society, May 2012.

[19] Martin Mulazzani, Philipp Reschl, Markus Huber, Manuel Leithner, Sebastian Schrittwieser, and Edgar Weippl. Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting, 2012.

[20] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless Monster: Exploring the Ecosystem of Web-based Device Fingerprinting. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 541–555, Washington, DC, USA, 2013. IEEE Computer Society.

[21] Toby Osbourn. The EU Cookie Law. `http://www.theeucookielaw.com/`. Retrieved May 9, 2014.

[22] Lalit Patel. JavaScript/CSS Font Detector. `http://www.lalit.org/lab/javascript-css-font-detect/`, March 2007. Retrieved April 6, 2014.

[23] Mike Perry, Erinn Clark, and Steven Murdoch. The Design and Implementation of the Tor Browser [DRAFT]. `https://www.torproject.org/projects/torbrowser/design/#fingerprinting-linkability`. Retrieved May 11, 2014.

[24] Arnold Roosendaal. Facebook tracks and traces everyone: Like this! `http://ssrn.com/abstract=1717563`, 2011.

[25] C. E. Shannon. Prediction and Entropy of Printed English. *Bell System Technical Journal*, 30(1):50–64, 1951.

[26] ThreatMetrix. Proxy and VPN Detection. `http://www.threatmetrix.com/technology/proxy-and-vpn-detection/`. Retrieved April 6, 2014.

[27] Ting-Fang Yen, Yinglian Xie, Fang Yu, Roger Peng Yu, and Martin Abadi. Host Fingerprinting and Tracking on the Web: Privacy and Security Implications. In *Proceedings of the 19th Annual Network & Distributed System Security Symposium*, February 2012.

Abbreviations

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CDP	Cisco Discovery Protocol
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DES	Data Encryption Standard
DHCP	Dynamic Host Configuration Protocol
DLL	Dynamic-Link Library
DOM	Document Object Model
EU	European Union
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
HTML	HyperText Markup Language
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
JSON	JavaScript Object Notation
LSO	Local Shared Object
NetBIOS	Network Basic Input/Output System
NPAPI	Netscape Plugin Application Programming Interface
OS	Operating System
OSI	Open Systems Interconnection
PHP	PHP: Hypertext Preprocessor
RGB	Red Green Blue
SMB	Server Message Block
SNMP	Simple Network Management Protocol
SQLite	Structured Query Language Lite
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
XUL	XML User Interface Language

Appendices

Appendix A

List of different HTTP browser requests

Mozilla Firefox 29.0:

```
GET http://uni.lu:80/ HTTP/1.1
Host: uni.lu
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:29.0)
            Gecko/20100101 Firefox/29.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
        */*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Apple Safari 7.0.3:

```
GET http://uni.lu:80/ HTTP/1.1
Host: uni.lu
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
        */*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2)
            AppleWebKit/537.75.14 (KHTML, like Gecko) Version/7.0.3
            Safari/537.75.14
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Opera 21.0:

```
GET http://uni.lu:80/ HTTP/1.1
Host: uni.lu
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
        image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2)
            AppleWebKit/537.36 (KHTML, like Gecko) Chrome/34.0.1847.132
            Safari/537.36 OPR/21.0.1432.57
Accept-Encoding: gzip,deflate,lzma,sdch
Accept-Language: en-US,en;q=0.8
```

Google Chrome 34.0:

```
GET http://uni.lu:80/ HTTP/1.1
Host: uni.lu
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
        image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2)
            AppleWebKit/537.36 (KHTML, like Gecko) Chrome/34.0.1847.131
            Safari/537.36
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
```

Microsoft Internet Explorer 11.0:

```
GET http://uni.lu:80/ HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: uni.lu
```

Appendix B

List of JavaScript attributes that *Fingerprint Privacy* detects and manipulates

- navigator.appCodeName
- navigator.appName
- navigator.appVersion
- navigator.battery
- navigator.connection
- navigator.geolocation
- navigator.javaEnabled
- navigator.language
- navigator.mimeTypes
- navigator.onLine
- navigator.oscpu
- navigator.platform
- navigator.plugins
- navigator.product
- navigator.userAgent
- navigator.buildID
- navigator.cookieEnabled

- `navigator.doNotTrack`
- `navigator.id`
- `navigator.productSub`
- `navigator.vendor`
- `navigator.vendorSub`
- `screen.height`
- `screen.width`
- `screen.colorDepth`
- `screen.availHeight`
- `screen.availWidth`
- `screen.pixelDepth`
- `Date().getTimezoneOffset()`

Appendix C

Fiches de suivi de stage

Université du Luxembourg

Faculté des Sciences, de la Technologie et de la Communication

Bachelor en Informatique (*professionnel*)

TRAVAIL DE FIN D'ETUDES

FICHE DE SUIVI DE STAGE

L'étudiant(e) :

NOM, Prénom	FERREIRA TORRES Christof
Numéro matricule	0111131947

L'établissement d'accueil :

NOM	Université du Luxembourg

Responsable local de stage :

NOM, Prénom	JONKER Hugo

Responsable académique de stage :

NOM, Prénom	MAUW Sjouke

Période :

Du	17.02.2014
Au	09.03.2014

Document à compléter et à envoyer (même non encore signé) par email au responsable académique, avec copie au responsable local, le plus tôt possible après la fin de chaque période. Une période correspond à trois semaines de stage.

Ce document complété et signé devra figurer en annexe du mémoire de fin d'études.

Travaux effectués pendant la période :

In the first three weeks I have spent a lot of time in doing research about what is actually browser fingerprinting in order to understand better the context and to deal better with the problem. Furthermore I tried to understand how such a browser fingerprint looks like, how stable such a browser fingerprint is and what kind of research has already been done in this field e.g. if there are already technologies out there which help you to preserve privacy regards browser fingerprinting.

Another task of my initial research was about data acquisition. I had to look up for data sources, which could provide me with statistics about the most commonly used browser configurations. I found some websites which provide global and local statistics about browser configurations and I succeeded in contacting the people who are behind the Luxembourgish government website « MySchool.lu » and got some statistical data regarding the browser configurations of their visitors.

In addition to my research and the data acquisition, I also started developing a first version of the plugin, which intercepts and observes the HTTP requests and the HTTP responses made by the browser. The plugin can also already manipulate HTTP request headers for example in order to delete the HTTP Referer header if the host and referrer are not from the same domain. Finally the plugin is also able to inject custom JavaScript code into the HTTP responses, in order to override certain JavaScript objects (e.g. navigator, window.screen, etc.) and spoof the values that they return.

(*Si pertinent*) Travaux prévus pour la période suivante :

In the continuation of my work, I will be focusing on how to keep web identities separate across different websites, this means to provide always the same fake fingerprint for the same website. I will try to reuse the “FireGloves” project and maybe in combination with “Private Tabs”, in order to achieve this separation of the web identities and additional privacy regarding cookies.

Furthermore in the upcoming weeks I have also planned to find a way on how to keep your web identities separate across those social media sharing buttons on websites, in order to prevent the social networking websites to track you.

Finally if there is still some time I will also try to work on a way to detect browser fingerprinting scripts in order to warn the user when a website that he is visiting is trying to fingerprint him.

	Le responsable local,	L'étudiant(e),
Date :	……	……
Signature :	……	……

UNIVERSITÉ DU LUXEMBOURG
Campus Kirchberg
6, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg
T. +352 46 66 44 52 17
F. +352 46 66 44 55 00
www.uni.lu

Université du Luxembourg

Faculté des Sciences, de la Technologie et de la Communication

Bachelor en Informatique (*professionnel*)

TRAVAIL DE FIN D'ETUDES

FICHE DE SUIVI DE STAGE

L'étudiant(e) :

NOM, Prénom	FERREIRA TORRES Christof
Numéro matricule	0111131947

L'établissement d'accueil :

NOM	Université du Luxembourg

Responsable local de stage :

NOM, Prénom	JONKER Hugo

Responsable académique de stage :

NOM, Prénom	MAUW Sjouke

Période :

Du	10.03.2014
Au	30.03.2014

Document à compléter et à envoyer (même non encore signé) par email au responsable académique, avec copie au responsable local, le plus tôt possible après la fin de chaque période. Une période correspond à trois semaines de stage.

Ce document complété et signé devra figurer en annexe du mémoire de fin d'études.

Travaux effectués pendant la période :

In the past 3 weeks I focused on the development of the plugin, mainly on the part of keeping web identities separate. I achieved this through creating an JavaScript object holding for each domain the user visits a fake fingerprint. This fingerprint is generated through a modified version of a random user-agent generator script.

Furthermore, I also implemented a way to block social plugins, by rejecting the requests going to Google, Facebook or Twitter. It's is currently more efficient than the «Disconnect» plugin, but as it's right now very restrictive, for example, it blocks also essential requests and blocks also requests where a user wants to voluntarily share something. This makes it not very user-friendly.

Moreover, I have worked on a way to track down third parties and to detect if a website is trying to read information useful for fingerprinting. In case of detection a user is warned and has to the choice to allow, thus whitelist the website or keep blocking the access and blacklist the website.

Finally, I started writing on my final report, by elaborating in the first two sections all the knowledge I gained across the past 6 weeks.

(*Si pertinent*) Travaux prévus pour la période suivante :

As a continuation to my current work, I will look for a way to persistently store the generated fingerprints in a whitelist and a blacklist, and give the possibility to the user to manage both lists.

I will try to reduce the restriction of the current blocking of social plugins in order to gain more usability and I am going to add the functionality to enable or disable social plugins for a given domain.

Additionally, I will improve the random fingerprint generator and the way users are notified up on the detection of possible fingerprinting websites.

Finally, I will continue updating my final report to the latest results and start writing the new sections about the current limitations of existing anti-fingerprinting techniques and about the development of my own plugin.

Le responsable local, L'étudiant(e),

Université du Luxembourg

Faculté des Sciences, de la Technologie et de la Communication

Bachelor en Informatique (*professionnel*)

Date : …… ……

Signature : …… ……

UNIVERSITÉ DU LUXEMBOURG

Campus Kirchberg
6, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg
T. +352 46 66 44 52 17
F. +352 46 66 44 55 00

www.uni.lu

Université du Luxembourg

Faculté des Sciences, de la Technologie et de la Communication

Bachelor en Informatique (*professionnel*)

TRAVAIL DE FIN D'ETUDES

FICHE DE SUIVI DE STAGE

L'étudiant(e) :

NOM, Prénom	FERREIRA TORRES Christof
Numéro matricule	0111131947

L'établissement d'accueil :

NOM	Université du Luxembourg

Responsable local de stage :

NOM, Prénom	JONKER Hugo

Responsable académique de stage :

NOM, Prénom	MAUW Sjouke

Période :

Du	31.03.2014
Au	20.04.2014

Document à compléter et à envoyer (même non encore signé) par email au responsable académique, avec copie au responsable local, le plus tôt possible après la fin de chaque période. Une période correspond à trois semaines de stage.

Ce document complété et signé devra figurer en annexe du mémoire de fin d'études.

Travaux effectués pendant la période :

I found a way to persistently store the web identities including the fingerprints, list of allowed/blocked social plugins and third parties through JSON. I create a popup dialog allowing the user to manage his web identities. However, the menu is currently complicated to understand and needs to be modified.

Furthermore, I reduced the restriction of blocking social plugins in order to gain more usability, for example, the previous code was not only block the Google+ button, but also Google Maps or Google fonts, the script simply blocked every request going to Google. This approach was very efficient but not the best solution. Social plugins can now be enabled or disabled individually per domain.

Moreover, I have added the functionality to remove ETags from HTTP headers, since these could be used to fingerprint a user.

Additionally, I improved the random fingerprint generator, by taking out Internet Explorer and improving the generation of random user-agents for Firefox, Chrome, Safari and Opera. Moreover, I have added time zone offsets, which are now also randomly generated. The script detects now properties called from the Screen object and the Date object in JavaScript.

I have added the possibility to individually allow or block browser plugins such as the Flash player, the QuickTime player or the VLC player for a particular domain.

Finally, I have updated my report regarding the latest progress of the development of my extension.

(*Si pertinent*) Travaux prévus pour la période suivante :

As a continuation, I will try to improve the menu for the management of the web identities, by adding the possibility to change individual attributes, regenerate individual attributes or web identities, remove web identities, allow or block certain attributes and copy web identities in order to link 2 web identities.

Additionally, I will try to improve in general the UI of my extension in order to become more user-friendly.

Furthermore, I will do some last improvements to the fingerprint generator, by adding the possibility to generate individual attributes and add the possibility to generate screen sizes and color depths, and HTTP accept headers.

Finally, I will try to finish Related Work, Methodology and Extension Development part of my final report.

	Le responsable local,	L'étudiant(e),
Date :	……	……
Signature :	……	……

UNIVERSITÉ DU LUXEMBOURG
Campus Kirchberg
6, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg
T. +352 46 66 44 52 17
F. +352 46 66 44 55 00
www.uni.lu