Fingerprinting for Web Applications: from Devices to Related Groups

Christine Blakemore

Thesis to obtain the Master of Science Degree in

Telecommunications and Informatics Engineering

Supervisors: Prof. Miguel Nuno Dias Alves Pupo Correia
Eng. João Vasco de Oliveira Redol

Examination Committee

Chairperson: Prof. Paulo Jorge Pires Ferreira
Supervisor: Prof. Miguel Nuno Dias Alves Pupo Correia
Member of the Committee: Prof. Maria Dulce Pedroso Domingos

May 2016

Dedicated to my family, who always supported me in every decision.

Acknowledgments

I would like to thank my supervisor Prof. Miguel Pupo Correia for his never ending support and enthusiasm, as well as my co-supervisor Eng. João Redol for all granted help, during the course of this work.

I warmly thank Prof. Miguel Pardal, Prof. Nuno Santos and Dra. Ibéria Medeiros for their comments on a preliminary version of the work.

To my family, Margarida Mendes, Craig Blakemore, Rosa Mendes, Abílio Mendes, Lurdes Mendes, José Mendes, Sofia Seromenho, Carolina Mendes and Constança Mendes, thank you all for your love and support, whom without I would have never been able to achieve this goal.

Lastly, I am grateful to my friends, for their support and companionship throughout my academic progress.

Thank you all!

Resumo

Identificar utilizadores e os seus dispositivos é tão importante em aplicações web como em vários outros contextos. A identificação de utilizadores em aplicações web geralmente envolve um processo de autenticação, por exemplo, com a inserção de um nome de utilizador e de uma senha. Esta identificação é também possível sem autenticação explícita, utilizando *cookies* ou *fingerprints* do dispositivo. O processo de *device fingerprinting* é útil para vários fins, como por exemplo para servir como um segundo fator de autenticação. Este trabalho pretende ser um passo em direção ao cross-device *fingerprinting*, isto é, a identificação do mesmo utilizador em diferentes dispositivos utilizando métodos de *fingerprinting*. No entanto, temos como alvo uma variante do problema a que chamamos de *related group fingerprinting*. Nós definimos um *related group* como um conjunto de pessoas (por exemplo, uma família) que partilham a mesma rede doméstica. Idealizou-se um esquema de *related group fingerprinting* que foi avaliado experimentalmente com dados de centenas de utilizadores. Esta avaliação sugere que *related group fingerprinting* é viável.

Palavras-chave: *Fingerprinting* na Web, Autenticação, Aplicações Web, *Cross-device fingerprinting*, *Cross-browser fingerprinting*, *Related Group Fingerprinting*

Abstract

Identifying users and user devices is as important in web applications as in many other contexts. In web applications, user identification usually involves an authentication process, e.g., providing a username and a password. Identification is also possible without explicit authentication using cookies or device fingerprints. Device fingerprinting is also useful for several purposes, e.g., to serve as a second factor of authentication. Recently some interest appeared in the problem of cross-device fingerprinting, i.e., of the identification of the same user in different devices using fingerprinting. We target a variation of the problem that we call *related group fingerprinting*. We define a related group as a set of persons (e.g., a family) that share the same home network. We devised a related group fingerprinting scheme that we evaluated experimentally with data from hundreds of users. This evaluation suggests that group fingerprinting is feasible.

Keywords: Web fingerprinting, Authentication, Web applications, Cross-device fingerprinting, Cross-browser fingerprinting, Related Group Fingerprinting

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Identifying users and user devices is as important in web applications as in many other contexts. In web applications, *user identification* usually involves an *authentication* process, e.g., providing credentials like a username, a password, or a token-generated code. Identification is also possible without explicit authentication using cookies or device fingerprinting.

Device fingerprinting, or web-based device fingerprinting, consists in gathering multiple pieces of information from the client's device and browser for identification purposes, e.g., the fonts and plugins installed, the model, version, and language of the browser, etc. [1, 2]. Device fingerprinting is useful for several purposes. It can serve as a second or third factor of authentication [3], complementing the above-mentioned credentials. It can also serve to detect lost or stolen user devices, when they contact a server that uses this form of authentication. Both examples are extremely important today, with the increasing number of phishing attacks and thefts of personal devices.

We use the term *fingerprint* to designate a set of features that identifies a device, browser, or user. To be useful, fingerprints have to combine information that allows to uniquely identify devices. The more diverse the feature's values, the more they may be unique, as their values may be less likely shared by multiple devices. Every time a user accesses a web page that includes fingerprinting software, the device fingerprint is collected and compared to a database of known devices. If the device is not in the database, it is added, increasing the number of known devices [4, 5].

1.1 Problem Statement

Recently some interest appeared in the problem of *cross-device fingerprinting*, i.e., of the identification of the same user accessing a website or web application from different devices (e.g., tablet, laptop, smartphone) without explicit authentication. However, this problem is challenging and may require collecting personal user data (e.g., mouse dynamics data), so we consider a variation of the problem that we call *related group fingerprinting*. We define a *related group* as a set of persons (e.g., a family) that share the same home network. Related group fingerprinting consists in detecting if a device belongs to a related group. We aim to solve this problem without using personal data.

In terms of applications, related group fingerprinting is useful for a spectrum of use cases. Usually there is

some level of trust inside related groups, so related group fingerprinting may be used to assign levels of risk to certain operations without explicitly knowing who the user is. For example, it may be used as an additional form of authentication in certain online operations, like accessing home banking services or school websites.

1.2 Methodology and Contributions

We did an experimental evaluation of this scheme with data from hundreds of users. First we developed two websites for volunteers to access. The websites contained both a short questionnaire and mechanisms to extract fingerprints. Then we studied mainly three aspects:

- Is it still possible to perform *device fingerprinting* given the very recent privacy mechanisms in web browsers that prevent the extraction of browser history data, one of the most powerful fingerprinting mechanisms [6]? We concluded that it is still realistic to perform device fingerprinting. There is also a contibution towards finding a minimum subset of features able to successfully perform device fingerprinting (Section 4.1).
- Given the fingerprints collected, is it possible to perform *cross-device fingerprinting*, i.e., to recognize if it is the same user in different devices? The answer was negative (Section 4.2).
- Is it possible to perform *related group fingerprinting*? The answer was, this time, positive as we wished (Section 4.3).

The *related group fingerprinting* scheme was designed based on the analysis of the experimental data obtained (see Section 3). For privacy reasons, this data does not include personal data like biometric data or browser history. Similarly to other fingerprinting schemes, each time a user accesses the website, a fingerprint is extracted and stored, in case the device fingerprint is not yet in the database. Moreover, the *group* of the device is also obtained and stored in case it had not been stored before, i.e., it is not in the set of groups of that particular device. A group is essentially a network to which a device is connected to. The group identifier is: the IP of the device of the used browser as observed by the server, in case the IP is owned by an Internet Service Provider (ISP); or the name of the organization obtained from the *whois* command, in case it is a company, governmental agency, etc. Notice that a device can belong to several groups, for example, if it is a smartphone connected to a home network, to a 4G network, and to an office network, which are 3 different groups.

The contributions of this dissertation are the following:

1. the definition of the problem of related group fingerprinting;
2. a scheme to perform related group fingerprinting;
3. an experimental study of this scheme;
4. an experimental study of device and cross-device fingerprinting.

1.3 Document Structure

In short, the remaining document is organized as follows. In Section 2 we describe multiple methods and concepts of previous work on user identification. Section 3 presents our approach as well as the used testing tool and metrics,

while Section 4 describes the experimental evaluation. Finally, Section 5 concludes the document, summarizing all performed work and challenges, also pointing out a few points in direction for future work.

Chapter 2

Background

In this section we summarize previous studies on multiple areas concerning user identification and the privacy issues that revolve around it, including detailed information about systems responsible for user concern.

In Section 2.1, we present the most common techniques used for user tracking purposes nowadays. Section 2.2 is dedicated to a more recent topic know as fingerprinting. In Section 2.3 we describe several techniques applied to user's behavior observation, from tracking a user's device location, to monitoring which web sites they attend on a daily basis, and finally by studying their interaction with their keyboard and mouse. Section 2.4 presents available protection systems or methods that may help in order to avoid being tracked on the web.

2.1 Current Mechanisms

2.1.1 IP Address and HTTP Referrer

One of the simplest ways of identifying a user and its location is via IP address. With an IP address, a website can calculate a devices approximate location. Nowadays it is likely that a computer shares an IP address with other devices in the same space (house or office). As IP addresses change regularly and may be assigned to different users, they fail as a good way to track a single user over time. In this sense, an IP address is problematic as a unique identifier and should be combined with other techniques in order to identify users. Users browsers inform advertising companies of which page is being viewed by sending the browsers HTTP referrer, and there are several situations where the user's information is revealed using this approach. For instance, every time a user clicks on a link, the browser which is being used loads the page and tells the website where the user came from. In other words, if a user clicks on link from a travelling site, for example www.travel.com, and this takes him to an outside website, this outside website is going to see the address of the www.travel.com page that it came from. This information is stored in the HTTP referrer header. Another case is when a web page includes an ad or tracking script.

2.1.2 Cookies

Websites providing free information or content depend on advertisement to continue operation. They rely on other websites to manage and run the advertisement on their page, turning to third party companies that serve

this advertisement. In order to serve websites their ads, companies resort to the use of cookies [7]. A cookie is a sequence of bytes that is created by the server and placed with the user's requests. Then, every time the client (browser) sends a request to the server, a cookie is sent. The cookie's data is transmitted inside an HTTP header, containing fields such as the Set-Cookie field and the Cookie field. The server sends a Set-Cookie (Set-Cookie: name=value) in the response header to set the field. Then, when there is a cookie set, the browser sends a Cookie (Cookie: name=value) in its request header. The Cookie header field value contains only the cookie value for a certain URL while the Set-Cookie header field specifies the cookie's value string [8]. When a user is browsing a given website, this same website generates the cookie, which is stored in a file belonging to the browser, and is next processed by the computers browser software. This cookie file is then stored in the browser's folder, containing the following attributes:

- *Expires and Max-Age:* The Expires attribute defines a specific date and time for when the browser should delete the cookie. In case the expiration date is not specified, the cookie will be deleted as soon as the page is closed. The Max-Age attribute can be used to set the cookie's expiration as an interval of seconds in the future, relative to the time the browser received the cookie.

- *Domain and Path:* The Domain and Path attributes define the scope of the cookie. They essentially tell the browser what website the cookie belongs to. A cookie can only be read, and therefore used, by the server who created it.

- *Secure and HttpOnly:* The Secure attribute is meant to keep cookie communication limited to encrypted transmission, directing browsers to use cookies only via secure connections. The HttpOnly attribute directs browsers not to expose cookies through channels other than HTTP (and HTTPS) requests.

Stored on the users' computers by web browsers, cookies allow a website to notice which users are returning. For instance, with cookies, websites are able to have information of when a user has already provided authentication information, a task considered useful by many users as they are then refrained from repeatedly providing their user names and passwords. Besides, web sites may even remember certain settings that users have previously personalized and intend on saving. However, the use of cookies results in plenty of issues regarding user's privacy, since they are in some cases aggressively used in the creation of profiles concerning their surfing habits. There are two kinds of cookies: session cookies and persistent cookies. Session cookies are temporary and only remain in the browser's cookie file until the user leaves the website. Persistent cookies, on the other hand, remain in the cookie file for much longer, depending on the cookie's lifetime [9]. By using cookies, websites are granted the ability to recognize returning customers and even keep track of a given sessions' state over HTTP protocol. Companies can monitor which sites their users visit, making it possible to catalog their tastes and interests, in order to build a dossier on their behavior. Since most common way of tracking web browsers nowadays is in fact via HTTP cookies, it lead to a growing awareness among users to what concerns their privacy, causing great discomfort [10]. As cookies have caught everyone's attention, several measures have been taken by the user community in order to respond to this threat. There are currently browser-extensions available that identify and reveal tracking [11, 12]. Also, modern browsers allow the rejection of all cookies and add the possibility of using a "Private Mode" or "Incognito Mode", where users may visit websites without leaving any kind of trace [13]. Supercookies are similar to regular cookies, except they cannot be erased by, for instance, using browser settings as "private"

or “incognito”, posing various security and privacy issues. They are stored in a different location from regular cookies, such as a file used by a plug-in, being Flash the most common. Supercookies are very hard to encounter, assuming the user is even aware of their existence. Adding to the difficulty in finding these supercookies, users have a hard time in erasing them, as they may have to dig through the file system in order to so. When a user has found them, he has to delete them manually. A few websites also use supercookies as a way to regenerate common cookies that a user has already deleted, also known as zombie cookies. With zombie cookies, a user’s previous cookie ID is linked with a newly assigned cookie ID [10]. However, when it comes to having various devices, the use of cookies fails. When swapping device, a user looses his previously stored login credentials, requiring new authentication in order to restore saved information. For instance, a user shopping on his telephone may want to later change to his tablet and continue with his shopping. When he decides to then change to the tablet, the shopping cart will be empty, forcing the user the re-authenticate to recover the saved contents. For instance, a particular user may be logged in a particular shopping website on his laptop, and once he tries to continue his shopping on his tablet, it requires new authentication in order to restore the shopping cart information.

2.1.3 Biometrics-Based Authentication

In the past years, the importance of reliability regarding user authentication has progressively increased, as user’s face now more threats. Current user authentication techniques, which involve the use of user IDs (identifiers, usernames for example) and passwords or PINs (Personal Identification Numbers), have many limitations, as they are easy to acquire illicitly. With an insecure authentication system, corporations face the worse consequences, including the loss of confidential information, denial of service and compromised data integrity. Once an intruder acquires a user’s identifier and respective password, he has full access to the user’s resources. There is no protection against repudiation. In general, even though data is sent over the web using secure protocols, current systems are not able of assuring that it is the credentials rightful owner initiating certain actions. One way to identify users and guarantee access control without the use of common authentication, with username and password, is through biometric authentication. The use of biometrics provides a more accurate and reliable user authentication method, where the concern is to identify a person based on his or her physiological or behavioral characteristics. Physiological characteristics are related with the human body, including fingerprint, face, retina, iris, and speech recognition. Behavioral characteristics however, are related to the pattern of a person’s behavior, which include typing rhythm, gait, and voice, for instance. As in any other system, when it comes to hackers, there are always ways at finding weak points in the system, vulnerable to attacks. With biometrics it is no exception. Password systems are prone to brute force dictionary attacks, where an attacker, typically running software instead of trying manually, tests all possible passwords systematically, beginning with words that have a higher possibility of being used [14]. It has the name “dictionary” attack as it refers to the attacker trying as many possibilities as the amount of words in an actual dictionary, exhausting hundreds or even millions of words in the attempt to determine the user’s password [15]. Although biometric systems do require more effort for this sort of attacks, there are several possible ones in this domain. These attacks may not be applied if there is some kind of supervision during authentication, although in different circumstances hackers may have the opportunity and time to attempt physical violation towards the client’s integrity. Biometric-based authentication has several assets, when it comes to usability, over more conventional systems as passwords. Biometrics are difficult to duplicate or steal, and almost impossible to share, although

their properties can in some cases be lost in serious accidents. One problem with this kind of authentication occurs when the data associated with a given biometric feature is compromised. With other authentication systems that use physical tokens, as cards or badges, these can simply be canceled and replaced with new ones. Another case, with usernames and passwords, these are effortlessly changed. With biometrics, when data is compromised, the user may quickly run out of features to use for authentication, as there are limited numbers of biometric features available. In Table 2.1 we can observe certain parameters as Performance, Cost, Acceptability and Circumvention associated with each biometric characteristics [16, 17].

Table 2.1: Comparison of Various Biometric Technologies.

	Performance	Cost	Acceptability	Circumvention
Face	Low	Low	High	High
Fingerprint	High	Low	Medium	Medium
Hand Geometry	Medium	High	Medium	Medium
Iris	High	High	Low	Low
Retina	High	High	Low	Low
Signature	Low	Medium	High	High
Voice	Low	Low	High	High

- *Performance:* Refers to the achievable speed and accuracy of the recognition, taking operational and environmental factors that may affect accuracy and speed into account.
- *Cost:* Represents the additional cost of including biometric technology in applications.
- *Acceptability:* Indicates the extent to which people are willing to accept using a particular biometric characteristic in their lives in a day-to-day point of view.
- *Circumvention:* Reflects the ease of how a system can be fooled with the use of fraudulent methods.

All the features that contribute to the strength of biometrics, and the fact that biometrics do not change over time, end up constituting its greatest weakness and liability. Once a set of biometric features has been compromised, it stays compromised forever.

2.2 Fingerprinting Mechanisms

Used as a mechanism towards users who limit cookies, fingerprinting arises as a hard to detect method that leaves no persistent evidence of tagging on user's computers. Web-based device fingerprinting is the process that collects information from a series of unique characteristics of the user's computers, through their web browser, in order to identify the device [1]. Device fingerprinting captures these browser element properties through JavaScript, Flash, or other plugins, and forms a nearly unique identifier – a fingerprint [2]. A fingerprint is nothing more than that collected information. It is a set of attributes, collected from the user's system, which combination of values is very likely unique for each device, forming a device identifier. These system attributes include: time zone, screen resolution, version of installed software, installed fonts, installed plugins, and enabled cookies. Some of these can be inferred from the HTTP headers alone, yet others must be collected using client-side JavaScript, sending this same information back to the server [2]. The more diverse the attribute's values, as the list of fonts, the more they

are considered identifying, as their values are not shared by many devices, which happens with the version of the operating system, for example. Fingerprints are used as identifiers in order to track the users' device on the web, granting the association between browsing sessions and even creating a link to the user's identity. Every time a user accesses a certain webpage that includes the use of fingerprinting software, the device fingerprint is collected and compared to a database of known devices. If the device is unknown, it is added to the database, otherwise it can be matched with its previous entry. This way, each time a user visits a monitored webpage the database will be augmented with the devices information [4, 5].

We consider that fingerprinting mechanisms can be classified in terms of what they aim to identify:

Browser Fingerprinting: The goal is to identify the web browser, or the pair (device, web browser) or, possibly, the trio (user, device, web browser).

Cross-Browser Fingerprinting: The goal is to identify the device independently of the used web browser, or possibly the pair (user, device).

Cross-Device Fingerprinting: The goal is to uniquely identify the user even when using different devices.

Fingerprinting is a very common method used by advertising and anti-fraud companies, as it allows overstepping the existing limitations of using cookies. Additionally, the use of fingerprinting allows companies to track users and their activities. Due to its stateless nature, which makes it hard to detect and almost impossible to opt-out, fingerprinting raises various privacy issues. Browser settings as "private" or "incognito" do not prevent fingerprinting, a measure taken by cookie-conscious users in order to perform privacy-sensitive operations [4]. In a similar way to cookies, fingerprinting can be used both in constructive or destructive ways. Constructively, it may be used to combat fraud, confirming if someone who is trying to login into a web-based server is in fact a legitimate user rather than an attacker who got hold of stolen login credentials. Also, it can be used to combat click fraud, which happens when someone displays an advertisement on their website in return for payment each time that ad is clicked. Then, by running an automated script or computer program, it imitates a legitimate user clicking on an ad, for the purpose of generating charge per click without having actual interest in the target of the ad's link. On the other hand, fingerprinting can be used to track users by monitoring which sites they visit, without their knowledge and consent, and without giving them an option to opt out. Besides being used for tracking, it can still be used by attackers in order to deliver exploits[13]. It has become so precise that it has made browsers privacy-protection measures become useless, as the more users try to hide their identity with the help of privacy-protecting extensions, the more they stand out in the end. The EFF (Electronic Frontier Foundation) has implemented a fingerprinting algorithm, Panopticlick, which anonymously logs the configuration and version information of user's operating system (OS), browser, and plugins, and compares it to a database of many other users' configurations [10]. The goal is to understand how identifiable users are, and evaluate the capabilities of Internet tracking and advertising companies who try to record user's activities. In this experience, 94.2% of the browsers using Flash or Java were considered unique in the sample used.

2.2.1 Browser Fingerprinting

Browser fingerprinting is commonly used as part of anti-fraud and advertisement systems, as it might avoid the need of authentication with credentials provided by the user and the use of cookies [4]. Similarly to cookies,

fingerprinting can be used both for legitimate and illegitimate ends. Legitimately, it may be used to combat fraud, confirming if someone who is trying to login into a web-based server is in fact a legitimate user rather than an attacker who got hold of stolen credentials. It can also be used to combat click fraud, i.e., illegitimate clicking of advertisements to increase payment [13]. Illegitimately, it may be used to track the actions of a person, violating his privacy in some sense.

JavaScript-Based

Much user data, such as fonts used or cookies, can be obtained using JavaScript, which is an object-oriented scripting language used for various purposes as interactive web contents, asynchronous communication and dynamic document content alteration [18]. Allowing dynamic content to be executed on web browsers, grants developers the ability of creating rich and interactive web interfaces. By enabling asynchronous communication with servers, it is possible to constantly update data without a page refresh. JavaScript provides APIs that grant access to device and browser-related properties. According to [4], JavaScript has become a powerful fingerprinting tool. It discloses sensitive information including user-agent, architecture, OS language, system time, and screen resolution. The two JavaScript APIs that have been mainly investigated and exploited for fingerprinting purposes are:

- *Navigator object:* that represents the properties of the device and browser environment (browser name and version, supported plugins and MIME types, and OS and browser architecture);
- *Screen object:* that contains information about the settings of the device's screen displaying the browser (screen resolution, and colour and pixel depth).

JavaScript has been used for fingerprinting in other ways, making use of performance [5], HTML5 canvas [19], and browsing history [6, 13]. An experiment conducted by Mayer in 2009 where 1328 web clients were fingerprinted, reported that approximately 96% were uniquely identified. To do so, he concatenated the contents of the navigator, screen, navigator.plugins (asks the browser what plugin types it supports) and navigator.mimeTypes (gets a list of all the MIME types supported by the browser), and then performed a hash of these contents [20]. In 2010, Eckersley fingerprinted just under half a million browsers. By adding fonts, time zones and the browser's Accept headers to the set of contents used by Mayer, the results were of roughly 94% unique fingerprints. Eckersley also proved that the installed list of fonts is one of the systems most identifying features. One can obtain this list of fonts in two ways: with browser plugins or JavaScript [4]. The scripting language of Flash provides APIs that include font discovering methods. These methods were originally meant to ensure the proper appearance of text by browser plugins, but they can also be used to fingerprint the system. The list of fonts may also be collected with JavaScript by measuring and afterwards comparing the dimensions of text rendered with different fonts. This approach rapidly discovers which fonts are present on the OS, even with a long list of fonts. On the other hand, fonts that are not so popular may not be detected.

In [5] Mowery et al. introduce two new browser fingerprinting techniques, both with the use of JavaScript, based on timing and performance patterns. The first technique does not depend on functionality but on performance differences between browsers, meaning that even when there is limited JavaScript functionality, this approach can still be used. The second technique allows the entries in the user's NoScript whitelist to be queried by attackers. NoScript may provide an extra protection for users as it blocks scripts from running automatically on most

websites. In order for the web content to be executed, the site hosting this content has to be considered trustful, therefore it has to be added to a whitelist. As there is less content being loaded once NoScript is enabled, the pages loading times are also improved. Users may customize their NoScript whitelists for their favourite web sites, as with JavaScript disabled most sites do not work well. This technique shows that by requesting scripts from domains and later inspecting if the scripts were in fact successfully executed, it is possible to deduct whitelists from the NoScript plugin, which can be used as an extra fingerprinting feature. To know if the scripts were indeed successfully executed, JavaScript objects in the global address space have to be searched. For the execution of these techniques the authors used a combination of 39 individual well-established JavaScript benchmarks, and generated a fingerprint from the runtime patterns. In total, the runtime for fingerprinting was high, with 190.8 seconds per user, caused partly by an intentional 800ms timeout between tests. With a similar approach, the authors of [21] obtained what were considered superior results:

- The runtime was proven faster: less than 200ms (instead of 190s).
- It can be implemented in a few hundred lines of JavaScript and is undetectable by the user.

Canvas Fingerprinting

Canvas fingerprinting is a type of browser or device fingerprinting technique that uses the browser's Canvas API to draw invisible images and extract a persistent fingerprint without the user knowing about it [22]. Depending on the operating system, font library, graphics card, graphics driver and the web browser, the same text can be rendered in several ways on different devices. Researchers find that there does not appear to be a way to block canvas fingerprinting automatically without there being false positives that end up blocking legitimate functionalities. The following steps indicate the basic flow of operations in canvas fingerprinting:

1. In every visit a web page receives, the fingerprinting script draws text with a font and size of its choice and adds background colours with the *fillText()* method.
2. Then, the script calls the *ToDataURL()* method from the Canvas' API, in order to get the canvas pixel data in *dataURL()* format.
3. At last, the script takes the hash of the text-encoded pixel data, using it as the unique fingerprint and combining it with other browser properties such as the list of plugins, fonts or the *userAgent* string. The *userAgent* is sent every time a user connects to a website, giving the website information about the user's browser and operating system.

The methodology presented in [22] is split into two particular steps. The first consists of the identification of canvas fingerprinting detection methods, and a web crawler development for exploratory crawls. A web crawler is a computer program that searches the web in an organized and methodical way with the purpose of web indexing. This step allowed a formal and automated method development based on early findings, as a crawler is able to copy all the pages it visits to process afterwards. The second step consists of the analysis method and detection of canvas fingerprinting. The return value of ToDataURL() and the arguments of the fillText() and strokeText() methods were logged, so that the strings drawn onto the canvas could be found. All function call logs were parsed and combined in an SQLite database allowing the crawl data of being analysed. Each time a user visited a site,

cookies, localStorage items, cache metadata, HTTP request and response headers, and request bodies were added to the SQLite database and false positives were removed. After examining the distinctive features of false positives and the fingerprinting scripts found in the initial experiments, the upcoming conditions were followed in order to eliminate false positives:

- Both *ToDataURL()*, *fillText()* and *strokeText()* method calls should be present, both coming from the same URL.
- The canvas image read by the script had to contain over one colour and its aggregate size had to be greater than 16x16 pixels, as imposing a 16x16 pixel size limit made it possible to filter scripts that read too few pixels to extract the canvas fingerprint efficiently.
- The image should not be requested in a lossy compression format (JPEG for example), as with a lossy compression the returned image may lose slight differences that are essential for fingerprinting.

With these checks applied, the false positive ratio for the 100,000 crawl results decreases to zero.

Plugin-Based

A plugin is a software component that can be embedded inside a web page and add specific abilities or functionalities to it, allowing interaction between web pages and third parties. In short, a plugin only affects a certain page in which it is placed. The use of plugins allows developers to acknowledge the capabilities of the running system (operating system version) and application (plugin version). Plugins also work with browser extensions, allowing them to perform certain tasks as blocking advertisement or adding applications inside the browser. Extensions affect the browser itself and not only the web page. They are included in the program and extend the functionality of existing software applications (as Firefox or Google Chrome). For instance, an extension might change the user's interface by simply adding a button or toolbar. Two examples of popular web browser plugins are Java and Adobe Flash Player (Flash). The Java plugin is used in the display of interactive web content through Java applets. A Java applet is no more than a Java program that (only with user permission) runs on the web browser. Java has become less and less popular, as multiple vulnerabilities keep appearing. Flash is used for displaying videos and games on the web browser. As well as JavaScript, Flash is used by companies to fingerprint user's environments. Flash has an API for enumerating the system's fonts, favouring fingerprinters that intent on collecting as much information on the user's system as possible. In addition to font extraction, fingerprinting companies use Flash to circumvent HTTP proxies that are set up by the user and get more specific information about the device, such as the OS kernel version. This is not only problematic from a privacy perspective, but also in terms of security, as a malicious web-server can launch an attack aiming not only a browser but also a specific kernel. Another API call behaving differently is the one revealing the user's screen resolution. For instance if a user is utilizing two monitors with a dual-monitor setup, Flash will report the width of the screen as the sum of the two individual screens, allowing a fingerprint service to detect this multi-monitor setup [13, 4].

Extension-Based

With more recent browsers, developers are able to perform a series of actions as adding new functionalities or removing unnecessary features, all by creating browser extensions. An extension is a program, typically written

in JavaScript, that allows adding functionality to the browser. Some extensions may change the user's interface by adding, for example, a toolbar. Browser extensions, as plugins, may also be used by fingerprinters to collect information about the user's browser. The popular browser extension NoScript [12] allows the execution of web objects that are found on the user's URL whitelist (trusted URL list). However, it has been proven that the NoScript plugin can be used for fingerprinting purposes [5]. The custom whitelisted URLs are able to be collected by requesting scripts from various domains: if the requested script is executed, then the URL is present in the whitelist. This enables the fingerprinter to create his own list of URLs and check whether the user has these URLs on his whitelist or not [4].

Header-based and Server-side

HTTP headers provide information such as HTTP Accept headers, *userAgent* string, etc. The *userAgent* string, for instance, provides data as name, version and platform on which the browser is running. This information about the user's browsers is sent to web servers with every HTTP request. These server-side requests are then easily fingerprinted and analysed, causing the user's web browser to be identified [4].

2.2.2 Cross-Browser Fingerprinting

The previous techniques may be designated as browser fingerprinting, in the sense that they aim to identify a browser running on a device. The problem of *device fingerprinting* or *cross-browser fingerprinting* is different because there may be several browsers installed on a device.[1] There is some preliminary work on doing such cross-browser fingerprinting that aims to identify a device irrespectively of the browser being used [24]. These techniques rely on features that have to be independent of the browser, for example: OS name and version, layout engine type and version, list of fonts, screen resolution, and time zone. That work has shown that it is possible to do cross-browser (or device) fingerprinting based on such features. It also shows that their scheme is resilient to the modification of one of the features, i.e., that the scheme still identifies devices correctly if one of the individual fingerprints changes.

To the best of our knowledge there is no work on cross-device or related group fingerprinting.

There has been some work on preventing device tracking and fingerprinting due to privacy concerns [25]. TrackingFree is an anti-tracking browser system that, instead of blocking the use of identifiers like cookies and flash files, isolates these identifiers into different units or browser principals, blocking third-party tracking [26]. PriVaricator uses a policy randomization process [27]. For offset measurements as offsetHeight, offsetWidth and getBoundingClientRect, instead of returning the original offset value, the proposed policy approach is to return, respectively, zero, a random number between 0 and 100, and the original offset value with $\pm 5\%$ noise. This approach both generates plausible offset values as well as creates enough noise to confuse fingerprinting. Bloom cookies use Bloom filters as a privacy-preserving data structure to provide a better trade-off between privacy, personalization, and network efficiency [28].

[1] There is a related problem called device fingerprinting that aims at doing active identification of computers by sending them packets, e.g., [23]. Our problem is different: to identify devices that access a web server.

2.3 Behavior-Based Mechanisms

2.3.1 Location Tracking

Current technological advances regarding wireless location tracking present unprecedented opportunities for monitoring users' movements. While such technology supports useful location-based services, privacy and security concerns seriously trouble users. These privacy concerns are due to the fact that previous systems kept databases with information concerning user's actual locations. When it comes to the user's devices, smartphones tend to be within reach, meaning their location is practically the same as their owner's location. To track the location of a smartphone, is the same as to track the location of whom it belongs to. As most users prefer their location not to be tracked, all mobile platforms see the device's location as sensitive information and recognize that it needs to be protected [29]. As such, applications require user permission to access, for instance, the phone's Global Positioning System (GPS). Location-based services have the following three classifications of Position Awareness, Sporadic Queries and Location Tracking [30].

- *Position Awareness* devices monitor user's positions, such as in-car navigation systems, where the information is only used internally. The service relies on the knowledge that the device has of its own location.

- *Sporadic Queries* apply to services where the user initiates the transfer of position information to an outside service provider (to find out which is the nearest hospital for example). This information only contains the user's current position.

- *Location Tracking* services are based on other parties (mobile service providers, for example) tracking the user by receiving frequent updates of the user's position. An example is to help GPS applications reroute the drivers' path.

To address users concerns, a location-support system for in-building, mobile, location- dependent applications called Cricket [31] was designed, helping clients' devices learn their location and decide whether to advertise this information and whom to. Using listeners that hear and analyse information form beacons spread all over the building, it allows applications running on mobile and static nodes to find out their location. A beacon is a small device that is attached somewhere within the geographic space is advertises and placed somewhere unobtrusive as a ceiling or wall. These beacons are used in combination with radio frequency (RF) and ultrasound signals in a decentralized and uncoordinated architecture, and their transmission schedules are random and independent. The used receiver decoding algorithm makes use of a minimum number of modes from different beacons in order to compute a maximum likelihood estimate of a location. The project involving Cricket showed that many location-dependent applications, as in-building active maps and device control, can effortlessly, or with plain manual configuration, be developed. With modern mobile platforms, there are applications that are able to read aggregate power consumption on users' mobile phones, by reading its ampere-meter over a period of a few minutes, allowing information on the user's location to be collected. This makes it easy for attackers to infer the target mobile phone's location as long as he has some knowledge on the general area in which the user moves. In [29], a machine learning method is used based on previously collected power consumption information in order to try and identify which routes are taken by the user. The authors start by studying three types of user tracking goals:

1. *Route distinguishability:* This is a classification problem in attempt to understand whether the attacker can tell which out of several possible routes the user is taking. New samples are classified based on a previously collected set of power profiles associated with known routes. A power profile is a time series, which is then compared with other time series. After each comparison, a score is assigned in order to select which route is most likely the correct one.

2. *Real-time motion tracking:* Assuming to know which route the user is taking, the goal is to track the mobile device during movement and find out if it is possible to localize the device along the route and track the device's position on the route in real-time. Reference power profiles are collected in advance for a specific route, and new power measurements are constantly received from an application installed on the device.

3. *New route inference:* Aims to understand if an attacker can identify an arbitrary route taken by the user in some given area, by assuming the attacker has previously measured all the power profiles of every short route the user has taken in that same area. On the contrary to 1) where the potential routes are, supposedly, known in advance, here the future potential routes are not explicitly known. However, the area in which the device moves in is known, although the amount of possible routes may be too big to pre-record each one. In order to overcome this problem, only the power profiles of all road segments within the area are pre-recorded, as every possible route will be a set of concatenated road segments.

For the performed experiments, real power consumption data had to be collected from smartphones devices along several different routes. The authors developed an Android application called PowerSpy that collected various measurements as signal strength, voltage, current, GPS coordinates, temperature, state of discharge, and cell id. The route distinguishability algorithm has several performed experiences. One of the best results was from a 43 profile experience, for 4 different routes (when different directions, as driving from point A to B and from point B to A, represents two different routes) of 19 kilometres each. This resulted in a 93% successful classification rate. With another dataset of only 13 profiles and two different directions along the same road, the achieved classification was 100%. In order to conduct an evaluation on a real-time motion tracking algorithm, a set of 10 training profiles plus an extra test profile were used. Real-time tracking is simulated by serially feeding samples to the algorithm as is they were being received from an actual application installed on the device. Initially, when there were few power samples, the estimation error was high, as the location estimation was extremely inaccurate. A couple of minutes after, the true location is locked. The estimation is quite precise up until 20 minutes. But as velocity starts to increase, it then starts to diverge, resulting in a large estimation error, which the authors defend of being easy to prevent. 80% of the estimation errors are less than 1 km.

2.3.2 Site Preferences

When obtaining user's web browsing histories, it is crucial to preserve the user's anonymity, as these personal preferences and histories reveal information about a particular user. The web has always been a source of security and privacy issues, where users expect to be able to browse free of harm, valuing their anonymity above many other features. Now more than ever, certain characteristics as speed, appearance and interactivity are other crucial aspects that a system is required of having. Therefore, long-term state has to be stored inside the client's browser. To hide this information from malicious attackers is not an easy task [32]. This is why it is important to understand the

consequences of web authors being able to determine the web sites users have visited, as a users' web browsing preferences are an identifying characteristic, as they relate to a user's interests. These preferences are in fact believed to be individual, as a fingerprint or retinal pattern. There are several methods to determine which particular web sites a user has visited considered reliable and effective. One approach is to query URLs in the browser's history using the CSS (Cascading Style Sheets) :visited history detection vector. CSS-based history detection was discussed as a potential threat to users' privacy in several analyses of Web browser security [6, 20]. It works by allowing the attacker to determine whether a particular URL has been visited by a users' browser by applying CSS styles and distinguishing between visited and unvisited links. For an attacker to collect history information, he must supply the client with a list of URLs to check and infer which of these URLs exist in the client's history by examining the computed CSS values on the client-side [33]. About 25% desktop user agents and an even higher proportion of mobile browsers are susceptible to this technique. Alternatively, a timing analysis, which allows the detection of items in a web browser's cache, is also a well-known approach. In [6] a large-scale analysis of web browsing histories is presented and studied. The authors also investigate if a user's browsing history, or the list of websites visited by the user, constitutes a unique fingerprint, which is able to identify and track the user. When testing 500 web pages, with a dataset of just under 370 000 web histories in total, a majority of users (over 69%) was detected as having a unique fingerprint. In other words, their browsing history is unique. The system architecture was divided in two phases. The first to detect "primary links", such as www.google.com, and the second to use this information to query for secondary URLs associated with the detected primary link. Adding to the history results, the system also collected the user's browser version information, IP address and test date. Side-information such as cookies, flash cookies or any other persistent storage technique to track users was not used. Olejnik et al. [6] concluded that besides most users (69%) have a unique browsing history, users for whom they detected at least 4 visited websites were successfully and uniquely identified by their browser histories in most of the cases (97%).

2.3.3 Keyboard and Mouse Dynamics

In order to verify an individual's identity, keystroke dynamics may be used, based on his typing rhythm patterns. Keystroke dynamics is the process of monitoring users keyboard inputs and studying the way they type as a way of recognizing and authenticating them based on common patterns in their typing rhythm [34]. Keystroke dynamics is not an intrusive method, making it easy and suitable to computer access security [35]. Generally, most biometrics are non-intrusive, simply requiring the placement of a finger, a look in the proper direction, or a statement said aloud. To design an automatic pattern recognition system, there are three steps: representation, extraction, and classification. The representation of the input data measures the pattern's or the recognizable object's characteristics. The extraction phase is often referred to as a feature extraction problem, where only a selected number of measurements from the input are considered, either because these selected features are enough to identify the user, or because by adding more features the computational complexity of the problem will increase. Finally, classification and identification involve determining an optimum decision procedure [34]. The process used in [36] to verify user's identity when logging in uses a modified login sequence. Adding to the user's login credentials (name and password), the system requires two extra strings: the user's first and last names. The authors believed that by typing familiar strings instead of a large number of unfamiliar ones, having still to read text before in order to do

so, the method becomes less error prone. The verifier achieved a rate of less than one percent for an imposter pass, whenever an imposter had the information of user login name, first and last names, and password. According to the authors of [35], researchers have presented studies in the past where they show that different individuals do in fact demonstrate that their typing characteristics are remarkably personal. They defend that those same characteristics can be successful when it comes to identification purposes. In [35], the authors collected profiles over a period of 7 weeks, where user's typing proficiency was not required, and where users ran the experiment on their own machines. Contrary to the previous experiment, the participants were asked to read and retype a series of phrases and/or simply type some sentences at that precise moment. Three classifiers were used in recognition: *Euclidean Distance Measure*, *Non-Weighted Probability*, and *Weighted Probability*. With the *Euclidean Distance Measure*, approximately 80% of the users were correctly recognized, based on samples derived from structured text. With the *Non-Weighted Probability* measure, approximately 85.6% of the users were correctly recognized. Performing better than both the previous classifiers, the *Weighted Probability* measure correctly recognized approximately 90% of users. In [34] the performance results were more extensive in comparison to the previous experience, as they were based on a database of collected profiles over a period of 11 months, for 63 users. Identically to the study performed in [35], user's typing proficiency was not required and user's ran the experiment on their own machines. They were asked to download and execute the experiment on their local machines and the results were then automatically sent back to the authors via email. The presented classification techniques were based on template matching and Bayesian likelihood models. The determined accuracy levels of the classifiers were in a range from approximately 83% to 92%, depending on which of the models was used.

Another biometric system, suitable for both intrusion detection and access control, is mouse dynamics. Requiring no special hardware device for data collection, mouse dynamics are collected passively and verified throughout the session. Mouse dynamics are the characteristics of a specific user's actions, received from the mouse input device, while interacting with a specific graphical user interface. In order to understand the actions received from the input device, the authors of [37] consider that identifying the categories in which these actions fall is crucial. Therefore, a mouse action can be classified as one of the following:

- *Mouse-Move (MM):* regular mouse movement;
- *Drag-and-Drop (DD):* action with the mouse button pressed, movement, then mouse button released;
- *Point-and-Click (PC):* regular mouse movement, followed by a click or double click;
- *Silence:* no movement.

Eight directions were considered for the mouse movement, one for every 45-degree area, numbered from 1 to 8. For example, direction 1 represents the all actions performed with angles between 0 degree and 45 degrees, while direction 2 represents those between 45 degrees and 90 degrees. There are factors that represent the components of what was termed as the mouse dynamics signature of a specific user. This signature may be used to verify user's identities. Some examples of the type of factors collected are: the average speed calculated for each distance travelled; the average speed calculated in each of the movement directions; and the average travelled distance for a specific period of time (with respect to different movement directions). A detection unit was designed before any validation experiment was conducted, using five users. By deploying the data collection software onto their

machines, users were asked to conduct their usual activities without restrictions. For each user, an average of 5,000 records were collected. None of these five users participated in the experiments described ahead. As a result, the parameters that proved to affect the detector's accuracy were the screen resolution, the operating system mouse pointer speed and acceleration settings, and the mouse button configuration. Then, three series of validation experiments were conducted. The main experiment, with 22 participants, reproduced real operating conditions in computing systems by allowing the participants to choose their operating environments and applications. This experience resulted in a total of 284 hours of collected raw mouse data, during 998 sessions, with an average of 45 sessions per user. The two remaining experiments involved only 7 participants, providing a basis for studying the confounding factors that rose from the main experiment by fixing the environment variables. The experiments resulted in a false acceptance rate (FAR) of 2.4649% and a false rejection rate (FRR) of 2.4614%. Although the experiment's results were not as expected, they were still confirmed to be better than the results of other well-established biometrics as voice and face recognition systems. In [38] the authors chose an approach that focuses on fine-grained angle-based metrics. These angle-based metrics have two advantages compared to previously studied metrics: they can recognize a user accurately with very few mouse clicks, and are relatively independent of the user's operating environment, becoming suitable for online re-authentication. The used system consists, mainly, of a recorder, responsible for gathering the mouse dynamics of a specific user, and a support vector machine (SVM) classifier, responsible for recognizing a user as either an impostor or an authenticated party. The conducted experiment gathered two data sets: one with only 30 users (under controlled circumstances) and another with over 1,000 users (on a forum website). The experiment, when tested for performance in terms of verification accuracy and time, resulted in an equal error rate (EER) of 1.3% with just 20 mouse clicks.

2.4 Protection Mechanisms

With the purpose of avoiding user tracking, researchers have proposed various solutions that target two of the third-party tracking steps: disabling third-party cookies to mitigate the unique identifier, and eliminating requests for private information like blacklisting known tracking servers. The *TrackingFree* system [26] is an anti-tracking browser system that instead of disabling unique identifiers, as cookies and flash files, it isolates these identifiers into different units or browser principals, blocking third-party tracking practices. This means the identifiers still exist, except they are not unique among different web sites. This way, third-party tracking web sites are not able to correlate the user's requests sent from several principals with those identifiers. *TrackingFree* is divided into two parts: isolation unit (web principal) and kernel. Each principal, that runs on the kernel, has its persistent storage isolated. This prevents third-party contents of different principals of sharing the same identifiers. Adding to this isolation method, content allocation is just as important. The principal manager determines, dynamically, how to put different frames into different principals based on the user's activities, frame properties and principal organization. Principal communication is also important, both for browser's privacy and compatibility, and is handled by the *message policy enforcer* (restricts the range of explicit communication for privacy-preserving purposes) and the *public history manager* (proposes a secure history sharing channel). This system also gives users flexibility to control the balance between anti-tracking and user experience. This system's evaluation showed that all the 647 trackers found on the Alexa's Top 500 web sites were blocked with affordable overhead. As browser-based

fingerprinting mechanisms have surfaced as an effective alternative to cookie-based tracking, privacy-enhancing technologies start to appear in order to work around browser fingerprinting. According to [27], the main problem with fingerprinting is the *linkability* of the fingerprint, and not its uniqueness, as fingerprints are hard to be linked across multiple website visits. In other words, it is a problem to link the same fingerprint across various subsequent visits. The authors propose a browser fingerprinting prevention method, the PriVaricator, which attempts to provide users with protection against *explicit fingerprinting*, as capturing browser environment details for example. PriVaricator uses a careful policy randomization process. For offset measurements as offsetHeight, offsetWidth and getBoundingClientRect, instead of returning the original offset value, the proposed policy approach is to return, respectively, zero, a random number between 0 and 100, and the original offset value ±5% noise. This approach was chosen as it could both generate plausible offset values as well as create enough noise to confuse the applied fingerprinting efforts. These policies are controlled by a *lying threshold* (θ), responsible for controlling how fast the PriVaricator starts lying (after how many accesses to the offsetHeight value, for example, does the policy kick in), and a *lying probability (P(lie))*, responsible for specifying the probability of lying. For the policy concerning the randomization of plugins, the authors define the probability *(P(plug_hide))* of hiding each individual entry in the browser's plugin list (whenever the navigator.plugins list is populated). To exemplify, with the configuration (*Rand_Policy* = Zero, θ = 50, *P(lie)* = 20%, *P(plug_hide)* = 30%), the PriVaricator is notified to start lying after 50 offset accesses, to lie in 20% of the cases, to return the value 0 when lying, and to hide approximately 30% of the browser's plugins. After experiments were executed, the authors found that PriVaricator proved itself able to deceive all of the used fingerprinters for a large fraction of tested combination settings. The authors also concluded that most fingerprinting providers derive fingerprints by following more complicated approaches than simply hashing fingerprintable attributes together.

Chapter 3

The Fingerprinting Approach

This section presents our approach. We start with Section 3.1, where we present our device fingerprinting scheme, as it is the basis of our related group fingerprinting scheme. Section 3.2 discusses how the problem of cross-device fingerprinting might be solved, and Section 3.3 presents our related group fingerprinting scheme. The developed testing tool and resulting datasets are presented in Section 3.4, followed by the chosen evaluation metrics in Section 3.5.

3.1 Device Fingerprinting

With the help of JavaScript we are able to extract multiple properties from user's devices. As previously mentioned in Section 2, the cross-browser fingerprinting method of [24] used features such as lists of installed fonts, OS version, screen resolution, etc. That work concluded that it was possible to create a unique fingerprint based on a set of these features and that, even if one of them changed, the scheme would still identify the device. This suggests that some of the fingerprints may change and, in fact, this is true for most of them. For instance, the installed fonts tend to change as more fonts may be installed, and the OS version tends to change due to system updates. Notice that we use the soft expression *tend to* in order to suggest that is what is expected, although changes may not in fact occur.

By developing a website with which users may interact, we run this same JavaScript script, gathering a set of data entries where each entry is a fingerprint. The first time a device contacts the website, an entry for that site with the fingerprints is stored in the server database. Our goals are to study if device fingerprinting is in fact possible with the collected features, as well as study the fingerprints' resilience and stability. We also intend on determining if we can find a minimum subset of features able to perform device fingerprinting successfully. These studies are featured in Section 4.1.

3.2 Cross-Device Fingerprinting

Cross-device fingerprinting is about identifying the same user using different devices. If cross-browser fingerprinting requires features that are independent of the browser, cross-device fingerprinting requires features that are

independent of the device (e.g., tablet, laptop). This is equivalent to say that the fingerprints have to be related with who the user is and how he acts. We can envisage three options:

- Fingerprints with configurations the users use across devices: our experience suggested that this would not work but we evaluated experimentally if it is possible to create fingerprints with such data (Section 4.3).
- Fingerprints with static biometric data (fingerprints from fingers, face geometry, iris, etc. [39] - Subsection 2.1.3): such biometric data is used for authentication so it would work, but we are interested in schemes that do not require the user to authenticate and that are not blunt violations of privacy, so we excluded them.
- Fingerprints with dynamic biometric data (keystroke dynamics [35], mouse dynamics [37], etc. - Subsection 2.3.3): this would be possible but it would require that the user interacted with the application for some time for collecting the fingerprint, making it application-dependent, not very practical, and problematic from the privacy angle, so we discarded it.

3.3 Related Group Fingerprinting

As already explained, the purpose of related group fingerprinting is to identify if a device belongs to a *related group*, i.e., to a group of persons who live together, e.g., a family. In practice, we define a related group as a set of persons that share the same home network. We use the term *group* to designate a set of devices that share the same company or organization network. A device may belong to several groups and related groups (e.g., a company network, a company network, and a home network).

We assume that home networks use private IP addressing, i.e., are behind a network address translation (NAT) router. Therefore, the identifier of a related group is the public IP address of the NAT router, which is the sender IP observed by the application's web server in the IP datagrams it receives from the device. Using for instance the *whois* command it is possible to understand if an IP address belongs to an ISP, so if the corresponding group is a candidate for being considered a related group. The identifier of a group that is not a home group is the name of the network, e.g., the content of the field *netname* returned by the *whois* command. The name cannot be an IP address, as the company or organization may have many public IP addresses.

The decision if a group is a *related group* can be based on one or more criteria, for instance:

1. *ISP-provided IP address:* a related group must have an IP address assigned by an ISP as observed in *whois*, as we do not envisage a related group having another kind of IP address;
2. *Limited number of devices:* a related group cannot contain more than D_{thresh} devices (e.g., 15), to exclude other networks with an IP owned by an ISP (e.g., the ISP network itself);
3. *Used mostly out of business hours:* accesses from the network are mostly made out of business hours (at night, during weekends), again to exclude other networks.

The related group fingerprinting scheme is represented in Algorithm 1. The algorithm uses criterion 1 and is executed by the server when a message with a fingerprint is received from a browser. The algorithm does not return anything, only adds data to the server database. The main types of queries to that database are the following:

Algorithm 1: Related group fingerprinting algorithm for collecting a fingerprint (executed by the server)

```
Input: HTTP request M received from the client
if M is the initial request from the client then
    add fingerprinting scripts to the reply;
else
    if M contains fingerprint then
        F ← get_fingerprint(M);
        if F ∉ database then
            G ← get_group(M);
            add_device(database, F, G);
        else
            add_group_to_device(database, F, G);
```

- Is group G a related group?
- To which related group(s) does the device with fingerprint F belong to?

There are a few pathological cases that deserve some discussion:

- What happens if the IP address of a home network with users accessing the web application changes? In that case, the server will keep data about two related groups, identified by the old and the new IP addresses, although the related group is actually the same. We do not envisage this to be problematic to most applications, if these events are not too frequent, which is the case in major ISPs.
- What happens if two different devices from unrelated persons have the same fingerprint? In that case the database will recognize the two as a single device that will belong to a set of groups that contains the groups of both devices. We do not expect this to be problematic if these collisions are rare.

3.4 Testing Tool

In order to test and analyse our fingerprinting mechanisms, we needed to find a way of collecting an experimental dataset from a group of users willing to contribute. We came up with the possibility of creating a website with which users could interact. They would access this website as a very simple and quick task, allowing us to collect all the necessary features for the fingerprint. Thus, for the experimental evaluation two different websites were developed, each destined for a different data analysis.

3.4.1 Website and Datasets

The first website (Website 1) aimed to interact with as many users as possible, so it collected both fingerprints and requested the users to answer a few questions. This experiment generated *Dataset I*. We later registered, for this same dataset, a total of 410 distinct users who willingly participated in the experiment. The second website (Website 2) had a different purpose, which was to study the fingerprint's stability and did not require users to answer any questions. Only 5 users participated in this experiment, that consisted in accessing that website for two weeks using the installed browsers on their devices. The data collected from this experiment resulted in our *Dataset II*. Both websites ran a JavaScript script responsible for collecting the information from the devices.

The website architecture has two main parts: Server Side and Client Side. Concerning the client-side part, it runs on the user's web browser, performing multiple HTTP requests to the server. We used Bootstrap [**?**], which is a framework to implement responsive design and allows faster and easier front-end (CSS and HTML) web development. The server-side part is the backend of the web application. It is responsible for the processing and the storage of the data provided by the user. PostreSQL [**?**] was used as backend relational database.

Table 3.1: Fingerprinting Features Collected in the Experiments

Reference	Feature	Description
Fp2	Plugins	Plugins installed in the browser, e.g., Java and Flash
Fp3	UserAgent	HTTP header field that identifies the browser and provides system details
Fp4	Browser	Browser used when accessing the website
Fp5	Cookies Enabled	True or false according to the cookies being enabled or disabled in the browser
Fp6	Display	Display's color depth, pixel depth, screen width, screen height
Fp7	System Fonts	List of installed system fonts
Fp8	Browser Language	Language preferred by the browser
Fp9	OS	Operating system in which the browser is running
Fp10	Time Zone	Time zone of the browser based on its geo-location data
Fp11	Touch	True or false whether the device has or has not touch screen
Fp12	IP Address	IP address of the device or its public address in case it is NATed
Fp13	Latitude	Latitude of the browser (only if user allows browser to provide the location)
Fp14	Longitude	Longitude of the browser (only if user allows browser to provide the location)
Fp15	HTTP Accept	HTTP header field that gives the accepted media types (for the HTTP response)
Fp16	HTTP Accept Encode	HTTP header field that gives the accepted content encodings
Fp17	HTTP Accept Language	HTTP header field that gives the set of preferred natural languages
Fp18	Platform	Platform of the browser, e.g., MacIntel or Win32
Fp19	Do Not Track	Do Not Track enabled or disabled in the browser

Table 3.1 shows the fingerprinting features obtained using the two experimental websites and a brief description (Fp1 is not shown as it designates the timestamp, that is not really a feature). Dataset I is composed of 531 fingerprints, whereas Dataset II has a total of 168 fingerprints collected from the above-mentioned 5 users. This means that we had a total of 699 complete accesses to the two websites. The data was collected in January and February 2016.

In order to perform the study we needed to know which fingerprints belong to the same user, therefore the websites asked for data that was then used to create a user ID. The purpose was to have data that uniquely identified

the user without his privacy being compromised. Therefore, we asked for two pieces of personal data reasonably harmless – the user's first name and the last 3 digits of his personal phone number – and calculated a SHA-256 hash of that data in the browser [40]. This hash results in the user ID and is sent to the server (unlike the name and digits that do not leave the browser at all), guaranteeing the confidentiality of the user's data and identity. This ID is used to compare and confirm results, identifying which results belong to the same user.

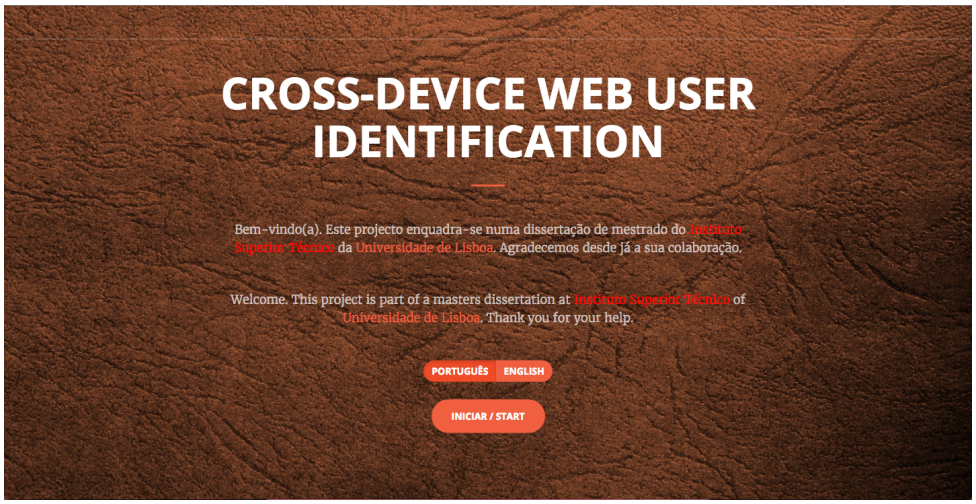

Figure 3.1: Website's First Page

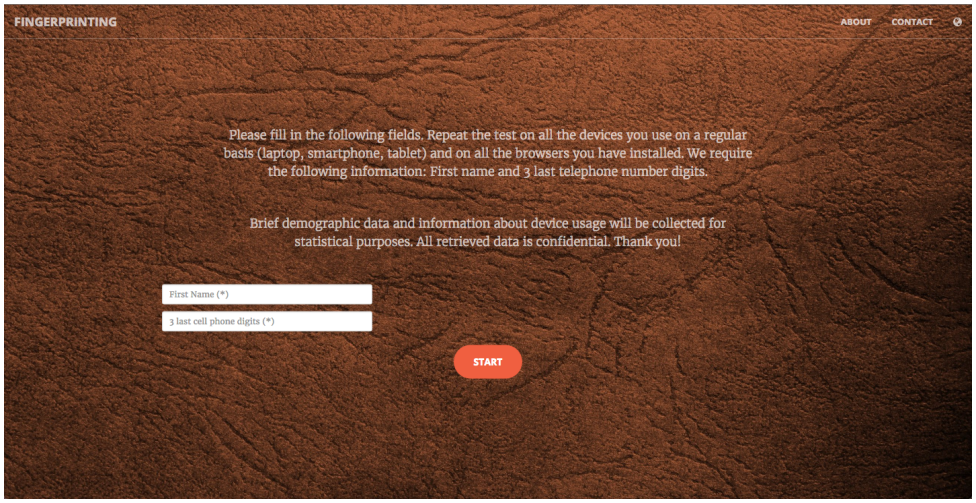

Figure 3.2: Website's Second Page

In Figure 3.1 we see how the front page of both websites looks like. It holds an introduction to the project and allows users to choose between the English and the Portuguese language. On the next page, shown in Figure 3.2, there are brief instructions of how the users are supposed to perform the test. It explains that it should be repeated on the user's multiple devices and browsers, and also informs the user about the confidentiality of the collected data. Finally, Figure 3.3 shows the page were the users are asked to answer the questions, stated in Table 3.3. The only difference between the two websites, is that Website 2 does not have the questions from Figure 3.3. Both websites also have the following tabs: Fingerprinting, with a very brief explanation of the matter; About, were users can read more about the project's goals; Contact.

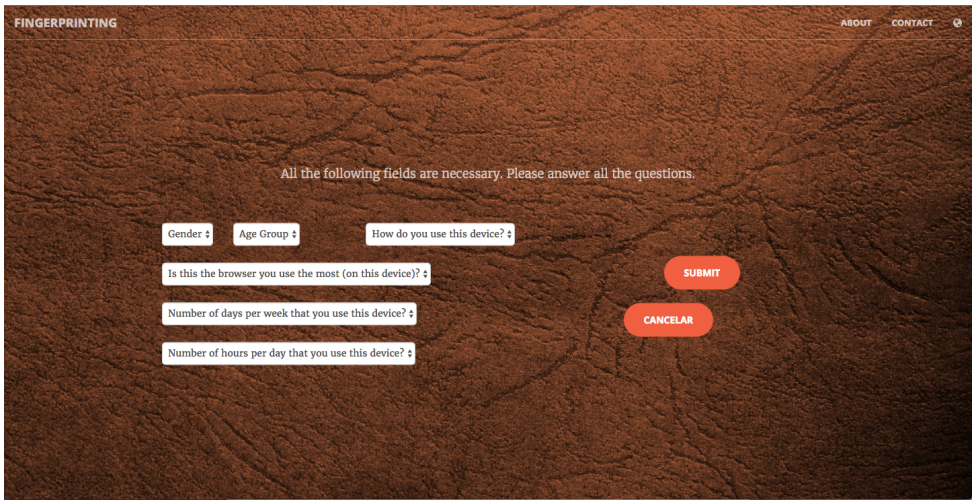

Figure 3.3: Website's Third Page

Out of the 531 entries of Dataset I, 410 different IDs were registered. In other words, 410 users participated in this experiment. Also, from a total of 531 data entries, 506 devices were classified as being unique, meaning that only 25 entries (4.71%) were repeated for a certain device (indicating a different browser). To study and analyze the retrieved user data, a testing tool was developed. This tool calculates parameters such as the Hamming distance between two data entries and the entropy values for each feature of the fingerprint.

In Table 3.2 we find three examples of dataset entries, with the corresponding collected features. Example 1 and 2 are for the same UserID, and indicate that the same device was used, but by accessing from two distinct browsers.

Table 3.2: Dataset Examples

Fingerprint Reference	Fingerprint Feature	Example 1	Example 2	Example 3
UID	UserID	1476041306a6e...	1476041306a6e...	035196bea0635...
Fp1	Timestamp	Sat Feb 06 2016 22:31:51 GMT-0200 (BRST)	Sat Feb 06 2016 22:33:40 GMT-0200 (BRST)	Sat Feb 13 2016 22:56:02 GMT+0000 (GMT Standard Time)
Fp2	Plugins	Shockwave, Flash, Acrobat Reader	Shockwave, Flash, Acrobat Reader, Java	Shockwave, Flash, Acrobat Reader
Fp3	UserAgent	mozilla/5.0,(macintosh— intel mac os x 10_11_1) applewebkit 537.36 (khtml, like gecko),chrome 47.0.2526.111 safari 537.36 opr 34.0.2036.50—MacIntel—pt-BR	mozilla/5.0,(macintosh— intel mac os x 10_11_1) applewebkit 601.2.7 (khtml, like gecko),version 9.0.1 safari/601.2.7 MacIntel—pt-br	mozilla/5.0,(windows nt 5.1) applewebkit 537.36 (khtml, like gecko) chrome 48.0.2564.109,safari /537.36—Win32—en-US
Fp4	Browser	Chrome 47	Safari 9	Chrome 48
Fp5	Cookies Enabled	TRUE	TRUE	TRUE
Fp6	Display	Color Depth: 24 Pixel Depth: 24 Screen Width: 1280 Screen Height: 800 Screen Available Width: 1280 Screen Available Height: 730	Color Depth: 24 Pixel Depth: 24 Screen Width: 1280 Screen Height: 800 Screen Available Width: 1280 Screen Available Height: 730	Color Depth: 24 Pixel Depth: 24 Screen Width: 1280 Screen Height: 800 Screen Available Width: 1280 Screen Available Height: 800
Fp7	System Fonts	Arial—Batang—Bauhaus,93—Bell MT Braggadocio—Calibri—Cambria,Math Cambria—Candara—Casual Chalkduster—Corbel—Courier,New Eurostile—Gulim—Harrington Hiragino Sans GB—Impact—Kino MT Lucida,Console—Lucida Sans Unicode MS Gothic—MS Mincho—MS PMincho MS Reference Sans,Serif—Matura MT Script Capitals—Menlo Microsoft Sans Serif—Microsoft Tai,Le—MingLiU MingLiU-ExtB—MingLiU_HKSCS MingLiU_HKSCS-ExtB—Mongolian,Baiti—Perpetua Plantagenet,Cherokee—Playbill—Rockwell SimHei—SimSun—SimSun-ExtB Times New Roman—Verdana	Arial—Batang—Bauhaus,93—Bell MT Braggadocio—Calibri—Cambria,Math Cambria—Candara—Casual Chalkduster—Corbel—Courier,New Eurostile—Gulim—Harrington Hiragino Sans GB—Impact—Kino MT Lucida,Console—Lucida Sans Unicode MS Gothic—MS Mincho—MS PMincho MS Reference Sans,Serif—Matura MT Script Capitals—Menlo Microsoft Sans Serif—Microsoft Tai,Le—MingLiU MingLiU-ExtB—MingLiU_HKSCS MingLiU_HKSCS-ExtB—Mongolian,Baiti—Perpetua Plantagenet,Cherokee—Playbill—Rockwell SimHei—SimSun—SimSun-ExtB Times New Roman—Verdana	Agency,FB—Arial Black—Arial—Bauhaus 93—Bell MT—Bodoni MT Bookman Old,Style—Broadway—Calibri Light—Calibri Californian FB—Cambria,Math—Cambria—Candara Castellar—Centaur—Century Gothic—Colonna MT Comic Sans,MS—Consolas—Constantia—Copperplate Gothic Light Corbel—Courier New—Engravers,MT—Eras Bold ITC—Forte— Franklin Gothic Heavy—Franklin Gothic Medium—French,Script MT— Georgia—Gigi—Goudy Old,Style—Haettenschweiler—Harrington Impact—Informal Roman—Kartika—Lucida,Bright— Lucida Console—Lucida Fax—Lucida Sans Unicode—MS Mincho— MS Reference,Sans Serif—Magneto—Marlett Matura MT Script Capitals—Microsoft Sans,Serif—Niagara Solid— Palace Script MTPalatino,Linotype— Papyrus—Perpetua—Playbill—Rockwell—Script MT Bold— Segoe UI,Light—Segoe UI Semibold—Segoe UI Showcard Gothic—Snap,ITC—Sylfaen—Symbol—Tahoma Times New Roman—Trebuchet MS— Tw Cen MT Condensed,Extra Bold—Verdana Vladimir Script—Vrinda—Webdings—Wide Latin—Wingdings
Fp8	Browser Language	lang=pt-BR—syslang=—userlang=	lang=pt-br—syslang=—userlang=	lang=en-US—syslang=—userlang=
Fp9	OS	Mac OSX,El Capitan—Unknown	Mac OSX,El Capitan—Unknown	Windows XP—32 bits
Fo10	Time Zone	-2	-2	0
Fp11	Touch	FALSE	FALSE	FALSE
Fp12	IP Address	189.26.138.232	189.26.138.232	2.82.122.136
Fp13	Latitude	-27.6074476	-27.60745094921905	39.4048977
Fp14	Longitude	-48.525459299999994	-48.5255664830993	-9.135404
Fp15	HTTP Accept	text/html,application/xhtml+xml application/xml—q=0.9,image/webp,*/*—q=0.8	text/html,application/xhtml+xml application/xml—q=0.9,*/*—q=0.8	text/html,application/xhtml+xml application/xml—q=0.9,image/webp,*/*—q=0.8
Fp16	HTTP Accept Encode	gzip, deflate, lzma	gzip, deflate	gzip,,deflate
Fp17	HTTP Accept Language	pt-BR,pt—q=0.8,en-US—q=0.6,en—q=0.4	pt-br	en-US,en—q=0.8
Fp18	Platform	navigatorPlatform: MacIntel	navigatorPlatform: MacIntel	navigatorPlatform: Win32
Fp19	Do Not Track	doNotTrack: unknown	doNotTrack: unknown	doNotTrack: unknown

3.4.2 User Statistics

Users who accessed the website had to answer a few questions for statistical purposes. Users were questioned about both personal information (e.g., age group, gender) and information concerning device/browser interaction. Figure 3.4 shows that over 60% of the participants were between 18 and 25 years old. We can also see that almost 3/4 of the participants were male, in Figure 3.5. Figures 3.6, 3.7, 3.8 and 3.9 show the percentage of answers for the rest of the questions, described in Table 3.3.

Table 3.3: Website Questions

Questions	Answers
Q1: How many days per week (average) do you use this device?	1-7 days
Q2: Number of hours per day that you use this device?	0 - 5 hours 5 - 10 hours 10 - 15 hours
Q3: How do you use this device?	Professionally Privately Both
Q4: Is this the browser you use the most (on this device)?	Yes No

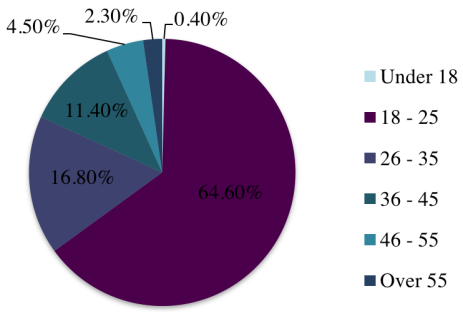

Figure 3.4: Participants Age Group

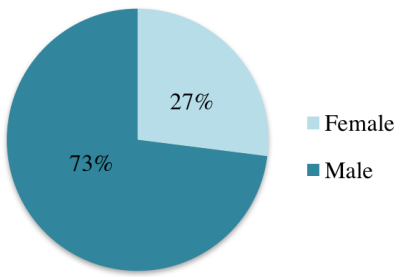

Figure 3.5: Participants Gender

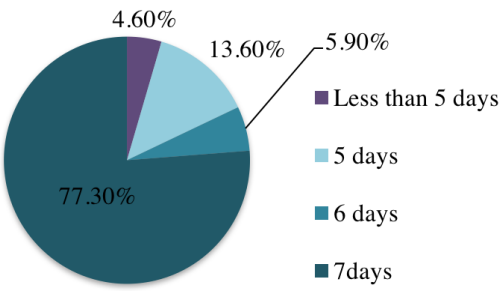

Figure 3.6: Question 1 (Q1)

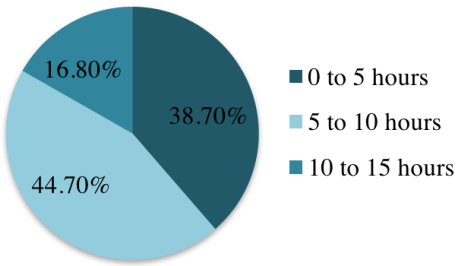

Figure 3.7: Question 2 (Q2)

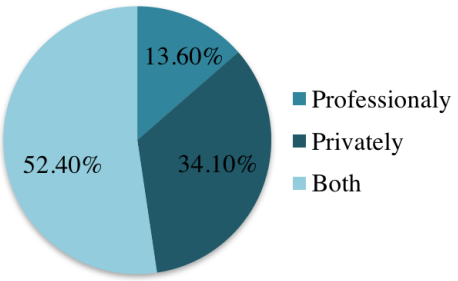

Figure 3.8: Question 3 (Q3)

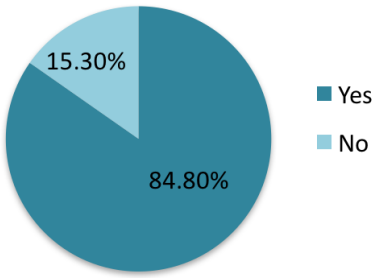

Figure 3.9: Question 4 (Q4)

3.5 Metrics

This section presents four metrics we use to analyze the experimental data: Hamming distance, entropy, precision, and accuracy.

3.5.1 Hamming distance

We use a generalization of the *Hamming distance* to measure the difference between two fingerprints, or the error in case they are two fingerprints of the same device [41]. The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different. For instance, with the words *ABCDE* and *ABCFF*, the Hamming distance is 2. We consider a fingerprint to be a vector of feature values (19 values in our case 3.1). For use with fingerprints, we generalized the metric so that what is compared are not symbols, but the feature values. For the collected 531 data entries, we created a 531x531 matrix where each position would have the calculated Hamming distance value between two different fingerprints. For example, for the following four fingerprints A=[Firefox], B=[Chrome], C=[Chrome] and D=[Chrome], we would have the following 4x4 matrix described in Table 3.4:

Table 3.4: Hamming Distance Between Fingerprints A, B, C and D

Hamming Distance	A Firefox	B Chrome	C Chrome	D Chrome
A Firefox	0	1	1	1
B Chrome	1	0	0	0
C Chrome	1	0	0	0
D Chrome	1	0	0	0

To calculate the number of *unique devices* using the Hamming distance metric using Table 3.4, these would be the following steps:

- First we calculate the total number of zeros, $\alpha = 10$ in this case.
- Then, subtract the number of zeros in the diagonal, which is the same as the number of fingerprints, which are 4. We now have $\alpha = 6$.
- As these fingerprints are repeated at least once, we have to divide α by two, leaving us with $\alpha = 3$.

- We use an auxiliary list called accessed_list. This list has *n* positions, corresponding to the *n* number of unique devices. Each position has the value of the number of times a certain fingerprint is repeated. In this case accessed_list = [1, 3], because we have two unique fingerprints, one that only accesses once, and one that does it three times. This list is only important for the cases in which a fingerprint accesses more than twice. In our case, only the second position of the list matters, and we have to subtract one more value for α, leaving us with α = 2.

- Final value of *unique devices* = 4 - α = 2 devices.

For our 531x531 Hamming distance matrix, there were the following steps:

- Total number of zeros: α = 589

- Subtract diagonal zeros: α = 589 - 531 = 58

- Divide by two: α = 58 $\div$ 2 = 29

- Here accessed_list has only four positions higher than 2, and they are all equal to 3. Meaning α = 29 - (3 - 2) $\times$ 4 = 29 - 4 = 25.

- Final value of *unique devices* = 531 - α = 506 devices.

3.5.2 Entropy

Once we have collected the necessary features, it is important to understand which will help obtain unique fingerprints. The *entropy* allows us to calculate how unique a certain feature is based on the amount of information it contains [10, 24, 42]. Using this metric it is possible to understand which combination of features produces unique fingerprints and those that are more useful to reach that uniqueness [42].

To calculate the entropy for each fingerprint feature, we use Shannon's entropy formula [43]. For a given feature f, the probability of ocurrence of the ith value ρ_i, and the number of fingerprints n (with $n = 531$ for Dataset I), the entropy H or the produced information is given by (in bits):

$$H(f) = -\sum_{i=1}^{n} \rho_i \times \log_2 \rho_i \tag{3.1}$$

For example, if there was a feature f' with different values for all 531 browser fingerprints of Dataset I, it would have the maximum entropy of $H(f') = 8.97$ bits. Furthermore, a fingerprint for that dataset might be composed of that single feature, as it would uniquely identify every browser and device.

To evaluate the importance of features in the fingerprint, we calculate their entropy for Dataset I. The higher the entropy the more unique the fingerprint is. Figure 3.10 shows the 10 features with the highest entropy values. The maximum entropy value corresponds to the IP address and is 8.52 bits, meaning that if we randomly choose a browser from the 531 browsers, it will share the same IP address with at most one browser in the other $2^{8.52} = 367$ browsers. As we can see in Figure 3.10, after the IP address, the user agent, display properties and fonts have the highest entropy values. However, entropy is not the single consideration to take into account when selecting features to compose fingerprint; e.g., the IP address is a dynamic feature so it is not adequate.

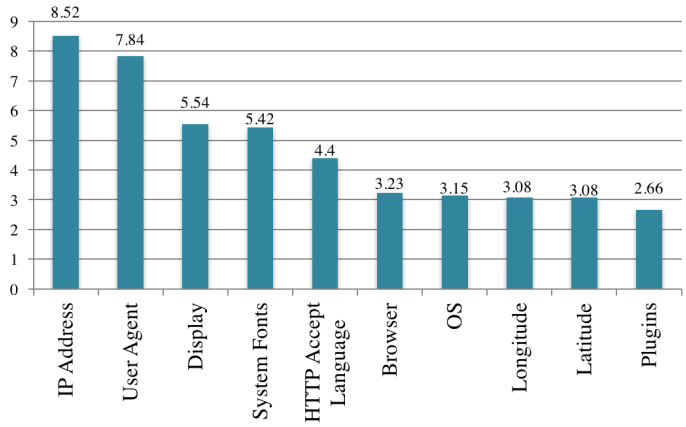

Figure 3.10: Features with Top 10 Entropy Values

The authors in [44] had a much larger dataset than our experiment, summing a total of 1,523 browser fingerprints and a maximum entropy value of 10.57 bits. The top 5 features with the highest registered entropy values were: plugins, system fonts, JavaScript fonts, mime Types, and User Agent. We can see a big difference between the entropy values of the plugins for example, and the reason for this difference may be due to the fact that our collected plugins do not have the version of the plugin. On the other hand, we found that both system fonts and the User Agent features are common to our studies and result in good identifiers, as they have high entropy values. In Table 3.5 we can find the calculated entropy values and the calculated number of unique values for each feature.

3.5.3 Precision and accuracy

The question we are interested in is essentially: *given two fingerprints F and F', does F' correspond to the same entity (device, browser, or user, depending on the case) than F?* We say that there is a *false positive* (FP) if the fingerprinting mechanism says the entity is the same (a positive P), that there is a match (Hamming distance is zero), and this is false. We say that there is a *false negative* (FN) if the mechanism says the entities are different (a negative N) and it is the same. In the case of *related groups* the relevant condition is not being the same entity but to belong to the related group, otherwise the idea is the same. Given these definitions we are interested in two metrics [45]. The first is the *precision*, which measures the confidence we can have when the mechanism says it is the same entity, there is a match. The second is the *accuracy*, which measures in some sense the correctness of the mechanism in terms of the rate between correct matches and the total (we abuse the notation and use TP to mean the number of true positives, P for the number of positives, etc.):

$$precision = \frac{TP}{TP + FP} \tag{3.2}$$

$$accuracy = \frac{TP + TN}{P + N} \tag{3.3}$$

Table 3.5: Entropy Values

Fingerprint Reference	Entropy Value (bits)	Unique Values (number)
Fp2	2.66	28
Fp3	7.84	330
Fp4	3.23	29
Fp5	0.04	2
Fp6	5.54	143
Fp7	5.42	129
Fp8	2.33	17
Fp9	3.15	17
Fp10	0.41	8
Fp11	0.96	2
Fp12	8.52	415
Fp13	3.08	155
Fp14	3.08	154
Fp15	1.18	4
Fp16	0.54	6
Fp17	4.40	80
Fp18	2.32	9
Fp19	0.88	6

Chapter 4

Experimental Evaluation

This section presents in detail the experiments we did in order to understand the problem and to assess our related group fingerprinting scheme. We analyse a few points on cross-browser fingerprinting in Section 4.1, cross-device fingerprinting in Section 4.2, an finally related group fingerprinting in Section 4.3.

4.1 Device Fingerprinting

For the device fingerprinting studies we intent to answer the following questions:

Question 1 – *Device Fingerprinting:* Is it possible to uniquely identify a certain device using only the features of Table 3.1?

This question is related to the challenge we mentioned of current privacy mechanisms in web browsers preventing the use of some of the most efficient features, namely browsing history [6]. Given these restrictions we limited the fingerprinting features to those of Table 3.1.

To determine whether it is possible or not to perform device fingerprinting with the static features, we have developed a program that calculates both the Hamming distance and the entropy values for each fingerprint feature. This same program also determines how many unique devices exist in the chosen dataset (Dataset I), and returns a list with these devices.

The program identified 506 unique devices, as there were 25 pairs of entries for which the Hamming distance was zero. Therefore, there were no false positives or false negatives. This means that the precision and the accuracy were both 1.

Question 2 – *Fingerprinting Resilience:* Does the fingerprint of a certain device tolerate changes in features?

This question is related to the issue of some of the features changing with time, e.g., the version of the browser changing. The objective is to understand if these changes still allow fingerprinting.

To test the fingerprinting resilience we used the same program from Question 1. The difference was in the used data sample, as we created 3 new datasets based on Dataset I. Each of these datasets is identical to Dataset I except that we removed one of the following features one at a time, respectively, System Fonts, OS, and Plugins. For the 3 datasets there were no false negatives (FN), as the mechanism never failed to identify two fingerprints

corresponding to the same device as being so. However, there were false positives (FP) as there was some confusion in identifying some of the devices.

The best results were found with the exclusion of the OS feature, with 506 unique devices, which corresponds to a precision and accuracy of 1. By excluding plugins, we also obtained a very good result, with 505 unique devices and 1 false positive, which means a precision of 0.998 and an accuracy of 1. Lastly, without system fonts, 491 unique devices were detected, 15 false positives, which gives a precision of 0.97 and an accuracy of 1.

This allow us to conclude that with our Dataset I the device fingerprinting mechanism tolerates well the change of one feature, as even the lowest precision is quite high (0.97). With changes to more than one feature results will be necessarily worse.

Question 3 – *Fingerprint Stability:* Are there any significant changes in the data over the days? Are the features more or less static?

With the purpose of studying the stability of the fingerprint, we used Dataset II, and tried to determine if it presented any changes over time. For the 16 devices present in this dataset, the only features that changed were the IP address and the geo-location (latitude and longitude), as some of the devices accessed the website from different networks and places. This is good as stability is needed for device fingerprinting and related group fingerprinting to work as expected.

Question 4 – *Minimum Subset:* What is the minimum subset of fingerprint features able to successfully identify a user in Dataset I?

Another interesting matter is to understand it we can find a suitable subset of features able to identify a user. For this we used Dataset I and the features with the highest entropy levels (Table 3.5), with the exception of the IP address, longitude and latitude (which are the most dynamic, making them bad candidates). Table 4.1 registers the number of unique devices and the precision for five different subsets of features, for a total of 531 data entries. Clearly the minimum subset depends on what is considered acceptable precision.

Table 4.1: Fingerprint Feature Subsets

Subset	Features in the Fingerprint	Unique Devices	Precision
1	Fp3, Fp6, Fp7	358	0.72
2	Fp3, Fp6, Fp7, Fp17	457	0.91
3	Fp3, Fp6, Fp7, Fp17, Fp4	487	0.96
4	Fp3, Fp6, Fp7, Fp17, Fp4, Fp9	487	0.96
5	same as above plus Fp2	505	1.00

4.2 Cross-Device Fingerprinting

In relation to cross-device fingerprinting we have a single question, as the answer is negative:

Question 5 – *Cross-Device Fingerprinting:* Using Dataset I, is it possible to perform cross-device fingerprint, i.e., to identify the same user behind two devices using only the features collected?

As referred in Section 3.2, cross-device fingerprinting is not an easy task. The distinguishing features would have to concern user behaviour (that we do not want to obtain) or uniform configuration across devices (that we

Table 4.2: ISPs and their Number of Related Groups

ISP	No. Related Groups
MEO Mobile	17
PT	95
NOS	101
Vodafone	87
Cabovisao	5
GVT	2
Total	307

did not expect to exist, but wanted to check). Recall that all entries in the datasets have a user identifier, so we know if two devices belong to the same user or not.

With the features we were able to collect, we cannot tell if two distinct devices do in fact belong to the same user. If two devices have different operating systems, for instance, the idea of these devices belonging to the same user is already rejected. This is the usual case as users have more than one device but they are different, e.g., a laptop running Windows and a tablet running Android. Therefore, the collected features are indeed insufficient to successfully perform a cross-device fingerprinting identification, which is a negative result.

4.3 Related Group Fingerprinting

This section evaluates the related group fingerprinting scheme presented in Section 3.3. For this purpose we enhanced both datasets with data about the IP owner obtained using the *whois* command. In this study we considered that a group is a related group using a single criterion of those we listed: the IP address being provided by an ISP or not (Criterion 1). The extended Dataset I emulates the database of the related group fingerprinting scheme.

Question 6 – *Related Groups:* What groups and related groups exist in Dataset I and what are their characteristics?

We start by distinguishing the related groups from the other groups by identifying IP addresses assigned by ISPs. We did this manually as more than 95% of the accesses were made from our country (Portugal) and it was trivial to identify the ISPs. Doing this at global level may require using a database of ISPs.

We identified 6 ISPs and 26 other networks. In the ISPs there were 307 individual IP addresses, which correspond to that same number of related groups applying only Criterion 1 (see Table 4.2), and a total of 333 groups. Notice that considering the 307 IP addresses to be related groups is a simplification that derives from the use of a single criterion to identify related groups; in reality some of these addresses may be from other networks. These addresses from other networks might be excluded using the other criteria of Section 3.3, but this would require a larger dataset.

In relation to the groups that are not related groups, there are three main ones (see Table 4.3). The major is UTL, our university (outdated, now ULisboa), which is not a surprise as we asked colleagues/students to access the websites, and INESC-ID is our research lab.

Question 7 – *Related Group Fingerprinting:* Is it possible to identify the devices of a related group? What is the precision and accuracy of our related group fingerprinting scheme?

Table 4.3: Groups that Are Not Related Groups

Group	No. Devices	No. Users
UTL	100	79
INESC-ID	12	9
PT Prime	12	12
Others	30	25

Table 4.4: Website Accesses for a User of Dataset II

Date	MacOS	Android1	Android2	Acc./day
Jan 21st	*1*	0	0	1
Jan 22nd	*2*	*1*	*1*	4
Jan 23rd	*1*	*1*	*1*	3
Jan 24th	*1*	*1*	*1*	3
Jan 25th	*2*	*1*	*1*	4
Jan 26th	*2*	*1*	*1*	4
Jan 27th	*2*	*1*	*1*	4
Jan 28th	*2*	*1*	*1*	4
Jan 29th	*2*	*1*	*1*	4
Jan 30th	*2*	*1*	*1*	4
Jan 31st	*2*	*1*	*1*	4
Feb 1st	*4*	*0*	*1*	5
Feb 2nd	*2*	*1*	*1*	4
Feb 3rd	*2*	*1*	*1*	4
Feb 4th	*4*	*2*	*2*	8
Total (devices)	31	14	15	

To start answering this question, we analyzed data from Dataset II for a user – one of the authors – with several devices and daily accesses between January 21st and Febuary 4th, 2016. Table 4.4 shows some data about these accesses, which were made from three different devices. When one of the devices made an access from the home network of the user, we put that entry and those below in italics to mean that the device was recognized as being part of the related group. This allows observing that after two days three of the user's devices were recognized as being part of the related group. After that exercise, we analyzed Dataset I looking for other users of the same related group. In fact we found two other users from the same related group (which we know to be persons from the same family in this case): (1) user that made an access with an iOS (iPad) device on Sat Feb 20 2016 18:42:15 GMT; (2) user that made an access with an Android device on Sat Feb 20 2016 18:50:43 GMT.

This analysis confirms that as expected it is possible to identify devices from the same related group.

We completed this analysis by inspecting the extended Dataset I where we identified all the related groups. As most devices ended up accessing the page only once, they ended up being excluded. This way, for a total of 85 users and 101 unique devices, Table 4.5 summarizes the data for 280 identified related groups. For each related group the table shows: a number we assigned to the related group (1st column); the ISP of the related group's home network (2nd); the number of devices belonging to the related group (3rd); the number of devices from this related group that made accesses from the home network (4th); the number of devices from this related group that made accesses from other networks (5th); the number of positives (P), that is equal to the value in the 4th column (6th); and the number of false negatives (FN), which are the devices from the related group that did not make accesses from the home network and therefor were not able to be identified (7th).

Given the data in the table we calculated the precision of the scheme for this data that was 1, and the accuracy

that was 0.999. We made one simplification that was to consider that the number of false positives (FP) is zero, thus the precision of 1; we do not have enough data to confirm this but we see no reason to believe there were false positives (devices wrongly assigned to a related group). On the contrary, the number of false negatives (FN) used was the one in the table that is pessimistic. In fact, as seen in Table 4.4, devices may not be added to the related group in the first contact, but they will eventually be included in the group.

Table 4.5: Related Group Analysis for Dataset I

Related Group No.	ISP	No. devices of RG	No. devices accessed home net.	No. of RG users w/accesses other nets	P	FN
1-15	MEO	1	1	0	1	0
16-17	MEO	2	1	1	1	1
18-78	NOS	1	1	0	1	0
79	NOS	1	1	1	1	0
80-86	NOS	2	1	1	1	1
87	NOS	2	1	2	1	1
88	NOS	3	1	2	1	2
89	NOS	5	1	3	1	4
90-106	NOS	2	2	0	2	0
107	NOS	2	2	0	2	0
108	NOS	3	2	1	2	1
109-110	NOS	4	4	0	4	0
111-180	PT	1	1	0	1	0
181	PT	1	1	1	1	0
182	PT	1	1	0	1	0
183	PT	1	1	0	1	0
184	PT	2	1	1	1	1
185-186	PT	2	1	1	1	1
187	PT	2	1	1	1	1
188	PT	3	1	2	1	2
189-194	PT	2	2	0	2	0
195	PT	2	2	1	2	0
196	PT	2	2	0	2	0
197	PT	3	2	1	2	1
198-199	PT	3	3	0	3	0
200	PT	4	4	0	4	0
201-242	Vodafone	1	1	0	1	0
243-244	Vodafone	1	1	0	1	0
245-247	Vodafone	2	1	1	1	1
248-250	Vodafone	3	1	2	1	2
251-252	Vodafone	5	1	3	1	4
253-267	Vodafone	2	2	0	2	0
268	Vodafone	3	2	1	2	1
269-270	Vodafone	3	3	0	3	0
271	Vodafone	3	3	1	3	0
272	Vodafone	6	3	3	3	3
273	Vodafone	11	6	4	6	5
274-278	Cabovisao	1	1	0	1	0
279	GVT	1	1	0	1	0
280	GVT	2	2	0	2	0

Chapter 5

Conclusions

We define and explore the problem of related group fingerprinting in web sites. We show that despite recent mechanisms that prevent access to browser history (a positive privacy measure), it is still possible to do device (or cross-browser) fingerprinting, at least with our main dataset of 500+ fingerprints. Related group fingerprinting leverages device fingerprinting and we have shown that it is possible to do it with high precision and accuracy. We also argue that cross-device fingerprinting can not be done using similar features.

5.1 Challenges

Throughout the project there were a few challenges that had to be overcome. The first has been mentioned before, and is the fact that the user's browsing history was not able to be collected. This is due to browser's most recent privacy restrictions. We expected to use browser history as a means for cross-device fingerprinting, so the impossibility of using this information led to a change of direction.

Another factor that had impact in the projects results, was the amount of users who participated in the experiment. Although the number of users was superior to what was initially expected, summing a total of 410 participants, we believe the results would have improved if this number were higher.

The instructions and website requested users to perform the experiment on every device used by the user. The purpose was to access the website once on every installed browser, on every device. Unfortunately, most users only accessed the website once, in other words, on one browser only. This definitely had impact on the results, as from 410 users, we only obtained a total of 531 data entries.

5.2 Future Work

The following directions are proposed as a development of the present research:

- Repeat the performed experiments for a larger number of users, and consequent dataset size. We believe that by increasing the dataset, the results may improve.
- Develop a cross-device fingerprinting technique with the help of biometric or behaviour-based solutions, with the user's consent. We concluded that cross-device fingerprinting was not possible with the chosen

approach, however it may be possible by adding user behaviour analysis into the evaluation.

- The related group fingerprinting may be further studied by trying to obtain deeper information about the related groups and their locations for instance. As mentioned before, the larger the dataset, the better the results, we believe.

Bibliography

[1] N. Nikiforakis, G. Acar, and D. Saelinger. Browse at your own risk. *IEEE Spectrum*, 51(8):30–35, 2014.

[2] D. Kim. Poster: Detection and prevention of web-based device fingerprinting. In *Proceedings of the 35th IEEE Symposium on Security and Privacy*, 2014.

[3] A. Bhargav-Spantzel, A. C. Squicciarini, S. Modi, M. Young, E. Bertino, and S. J. Elliott. Privacy preserving multi-factor authentication with biometrics. *Journal of Computer Security*, 15(5):529–560, 2007.

[4] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel. FPDetective: dusting the web for fingerprinters. In *Proceedings of the ACM SIGSAC Conference on Computer & Communications Security*, pages 1129–1140, 2013.

[5] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham. Fingerprinting information in javascript implementations. In *Proceedings of the Web 2.0 Security and Privacy Workshop*, 2011.

[6] L. Olejnik, C. Castelluccia, and A. Janc. Why Johnny can't browse in peace: On the uniqueness of web browsing history patterns. In *Proceedings 5th Workshop on Hot Topics in Privacy Enhancing Technologies*, 2012.

[7] D. M. Kristol. Http cookies: Standards, privacy, and politics. *ACM Transactions on Internet Technology*, 1 (2):151–198, 2001.

[8] A. Barth. Http state management mechanism. IETF Request for Comments: RFC 6265, 2011.

[9] S. Fiebrandt. What are cookies? what are the differences between them (session vs. persistent)? `$http://www.cisco.com/c/en/us/support/docs/security/web-security-appliance/117925-technote-csc-00.pdf$`, 2014. URL `http://www.cisco.com/c/en/us/support/docs/security/web-security-appliance/117925-technote-csc-00.pdf`.

[10] P. Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies*, pages 1–18. Springer, 2010.

[11] Ghostery homepage. `$https://www.ghostery.com/en/$`. URL `https://www.ghostery.com/en/`.

[12] Noscript. `$https://noscript.net/$`. URL `https://noscript.net/`.

[13] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 541–555, 2013.

[14] N. K. Ratha, J. H. Connell, and R. M. Bolle. Enhancing security and privacy in biometrics-based authentication systems. *IBM Systems Journal*, 40(3):614–634, 2001.

[15] E. H. Spafford. Opus: Preventing weak password choices. *Computers & Security*, 11(3):273–278, 1992.

[16] A. K. Jain, A. Ross, and S. Prabhakar. An introduction to biometric recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(1):4–20, 2004.

[17] S. Pankanti, R. M. Bolle, and A. Jain. Biometrics: The future of identification. *Computer*, 33(2):46–49, 2000.

[18] G. Richards, S. Lebresne, B. Burg, and J. Vitek. An analysis of the dynamic behavior of javascript programs. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 1–12, 2010.

[19] K. Mowery and H. Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In *Proceedings of the Web 2.0 Security and Privacy Workshop*, 2012.

[20] J. R. Mayer. Any person... a pamphleteer": Internet anonymity in the age of web 2.0. *Undergraduate Senior Thesis, Princeton University*, 2009.

[21] M. Mulazzani, P. Reschl, M. Huber, M. Leithner, S. Schrittwieser, E. Weippl, and F. Wien. Fast and reliable browser identification with javascript engine fingerprinting. In *Proceedings of the Web 2.0 Security and Privacy Workshop*, volume 5, 2013.

[22] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 674–689, 2014.

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

[24] K. Boda, Á. M. Földes, G. G. Gulyás, and S. Imre. User tracking on the web via cross-browser fingerprinting. In *Proceedings of the 16th Nordic Conference on Information Security Technology for Applications*, pages 31–46. Springer, 2012.

[25] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 227–238, 2014.

[26] X. Pan, Y. Cao, and Y. Chen. I do not know what you visited last summer: Protecting users from third-party web tracking with TrackingFree browser. In *Proceedings of the Network & Distributed System Security Symposium*, 2015.

[27] N. Nikiforakis, W. Joosen, and B. Livshits. Privaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web*, pages 820–830, 2015.

[28] N. Mor, O. Riva, S. Nath, and J. Kubiatowicz. Bloom cookies: Web search personalization without user tracking. In *Proceedings of the Network & Distributed System Security Symposium*, 2015.

[29] Y. Michalevsky, G. Nakibly, A. Schulman, and D. Boneh. Powerspy: Location tracking using mobile device power analysis. *arXiv preprint arXiv:1502.03182*, 2015.

[30] M. Gruteser and X. Liu. Protecting privacy in continuous location-tracking applications. *IEEE Security & Privacy*, 2(2):28–34, 2004.

[31] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proceedings of the 6th ACM Annual International Conference on Mobile Computing and Networking*, pages 32–43, 2000.

[32] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Protecting browser state from web privacy attacks. In *Proceedings of the 15th International Conference on the World Wide Web*, pages 737–744, 2006.

[33] A. Janc and L. Olejnik. Web browser history detection as a real-world privacy threat. In *Computer Security–ESORICS 2010*, pages 215–231. Springer, 2010.

[34] F. Monrose and A. D. Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation Computer Systems*, 16(4):351–359, 2000.

[35] F. Monrose and A. Rubin. Authentication via keystroke dynamics. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 48–56, 1997.

[36] R. Joyce and G. Gupta. Identity authentication based on keystroke latencies. *Communications of the ACM*, 33(2):168–176, 1990.

[37] A. A. E. Ahmed and I. Traore. A new biometric technology based on mouse dynamics. *IEEE Transactions on Dependable and Secure Computing*, 4(3):165–179, 2007.

[38] N. Zheng, A. Paloski, and H. Wang. An efficient user verification system via mouse movements. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 139–150, 2011.

[39] B. Miller. Vital signs of identity [biometrics]. *IEEE Spectrum*, 31(2):22–30, 1994.

[40] H. Gilbert and H. Handschuh. Security analysis of SHA-256 and sisters. *Selected Areas in Cryptography*, pages 175–193, 2003.

[41] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[42] M. Vaites. The effectiveness of a browser fingerprint as a tool for tracking. Master's thesis, Open University, UK, 2013.

[43] C. E. Shannon. Prediction and entropy of printed english. *Bell System Technical Journal*, 30(1):50–64, 1951.

[44] A. Faiz Khademi. Browser fingerprinting: Analysis, detection, and prevention at runtime. Master's thesis, Queen's University, Kingston, Canada, 2014.

[45] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.