RUHR-UNIVERSITÄT BOCHUM
Horst-Görtz-Institute for IT Security

Technical Report TR-HGI-2015-002

On the Effectiveness of Fingerprinting Mobile Devices
Investigating Modern Web-Tracking Mechanisms

Thomas Hupperich[†], *Davide Maiorca*[*], *Marc Kührer*[†], *Thorsten Holz*[†], *Giorgio Giacinto*[*]

[†]Chair for Systems Security, Horst-Görtz-Institute for IT-Security, Ruhr-Universität Bochum
[*]Department of Electrical and Electronic Engineering, University of Cagliari

Ruhr-Universität Bochum
Horst-Görtz-Institute for IT-Security
D-44780 Bochum, Germany

TR-HGI-2015-002
31 December 2015

On the Effectiveness of Fingerprinting Mobile Devices
Investigating Modern Web-Tracking Mechanisms*

Thomas Hupperich†, Davide Maiorca*, Marc Kührer†,
Thorsten Holz†, Giorgio Giacinto*

Abstract

Client fingerprinting techniques enhance classical cookie-based user tracking to increase the robustness of tracking techniques, e.g., in cases where a user regularly deletes cookies. A unique identifier is created based on characteristic attributes of the client device and then used for deployment of personalized advertisements or similar use cases. Whereas fingerprinting performs well for highly customized devices (especially desktop computers), these methods often lack in precision for highly standardized devices like mobile phones and tablets.

In this work, we show that widely used techniques do not perform well for mobile devices yet, but that it is possible to build a fingerprinting system for precise recognition and identification. We evaluate our proposed system in an online study and verify its robustness against misclassification.

Fingerprinting of web clients is often seen as an offence to web users' privacy as it usually takes place without the users' knowledge, awareness, and consent. Thus, we also analyze whether it is possible to outrun fingerprinting of mobile devices. We investigate different scenarios in which users are able to circumvent a fingerprinting system and evade our newly created methods.

1 Introduction

Tracking is an integral technique in today's Web and use cases reach from session management over personalization (especially related to advertisements) to fraud detection techniques. The "classical" approach for web tracking is based on HTTP cookies, where the web server stores some information on the client side in a persistent way that outlasts the current browsing session. Furthermore, other kinds of (transient or persistent) cookies are possible; Flash content or HTML5 storage are just two of many examples for technical approaches to tracking. Such tracking induces concerns related to the privacy of web users, especially since tracking usually takes place without the users' knowledge, awareness, and consent. Thus, users are inclined to delete cookies or perform other actions to avoid tracking.

*This technical report is an extended version of the paper published at FIXME [?].

To complement these state-based tracking techniques, stateless tracking (also known as *fingerprinting*) has thus become an increasingly important technique. In recent years, many features and methods have been proposed [12, 16, 20, 23] that facilitate the generation of fingerprints for device identification. The usual procedure works in two steps: first, characteristic attributes of a system are collected. In the second step, a unique identifier is derived from this input, and the resulting identifier is then used to recognize this single system. The attributes yield descriptive information about the system, e. g., specifications or customizations, and give clues about the kind of usage and configurations. The combination of such attributes—also called *features*—is desired to be as unique as possible, while weights can be leveraged to express the attributes' importance. The term *fingerprint* describes the set of attribute values as well as the resulting identifier for a client device.

Recent studies demonstrate that fingerprinting works well for highly customized devices (especially desktop computers), while lacking precision for highly standardized devices like mobile phones [12, 23]. This is due to the fact that fingerprinting relies on characteristic features that can be either customized by a user (e.g., installed fonts) or depend on the actual device (e.g., screen resolution or color depth). Mobile phones lack such customizations and thus tracking of such devices is still an open problem in practice. Due to the fact that mobile devices (especially Android-based mobile phones) significantly gained market share in the last years, tracking of such devices is evermore relevant.

In this work, we focus on this problem and examine it in detail. In a first step, we perform a comprehensive analysis of tracking libraries used in the wild. More specifically, we study nine commercial tracking libraries and analyze the fingerprinting techniques used by these commercial vendors. We find a wide variety of potential tracking methods and study the information gain provided by each feature. This analysis is based on a real-world data set consisting of data collected from more than 15.000 client systems. The main finding is that the currently used features do not perform well for mobile devices, especially due to the fact that such devices cannot be customized easily as compared to desktop computers. As such, we confirm the observation by Nikiforakis et al. [23] that tracking of mobile devices is a hard problem in practice.

In the second step, we propose several features that tracking systems could leverage to fingerprint mobile devices. We study four different categories of features (i.e., browser, system, hardware, and behavioral attributes) and discuss how they can be utilized for tracking. We implemented the proposed features, built a prototype of a fingerprinting system, and evaluate its effectiveness with 724 mobile users who took part in our experiments over a duration of 4 months.

As a third step, we study the robustness of our algorithm against *evasion attacks*, i.e., we study how a user could influence the features by changing device attributes to bypass our tracking system. Based on a discussion of the changeability of features, we evaluate four different evasion scenarios (e.g., using a second browser or a proxy connection). We find that users can evade fingerprinting, but that it is not as easy as one would expect at a first glance.

In summary, we make the following contributions:

- We provide a comprehensive analysis of existing tracking techniques used by (commercial) tracking companies and study the performance of such techniques for mobiles devices in a field study.

- We discuss how tracking for mobile devices can be improved. To this extent, we propose a fingerprinting system based on known and new features that result from a systematic study of browser, system, hardware, and behavioral attributes.
- We implemented the proposed system and evaluate the prototype in an online study with 724 participants, of which 459 participants accessed the experiment more than once over a duration of 4 months.
- We study evasion techniques against fingerprinting systems, i.e., we analyze how a user can bypass the tracking system by changing (some of) the features of her mobile device. We study the robustness of our proposed approach by evaluating evasion attacks under four different scenarios.

2 Study of Existing Fingerprinting Techniques

The main purposes of fingerprinting is to identify or recognize a device (or system). Use cases include fraud prevention as well as user tracking for personalized advertisement. In the Web, device recognition is often achieved by cookies which contain a unique identifier token for recognition. But modern Web techniques like JavaScript or HTML5 can also be utilized for this task. Modern methods are able to access a system's resources and obtain system specific information which can then be combined to the system's fingerprint. In general, a fingerprint is a set of characteristic attributes (also called features) which can be used to distinguish a specific system from other systems. In practice a fingerprint is often represented by a hash value over all features, but the plain list of features already satisfies the definition of a fingerprint. In contrast to cookies, the fingerprint of a device or system yields information about its specifications which enables meta-analyses of system groups. Instead of just recognizing unique systems, we can learn about different kinds and configurations of systems.

Given this definition, we review the approaches of real-world tracking libraries and then dive into technical details of state-of-the-art fingerprinting methods and their effectiveness.

2.1 Commercial Tracking Libraries

We analyzed nine known commercial tracking libraries, namely BlueCava, Device Ident, Inside Graph, Iovation, Threat-Metrix and many more. The following attributes were found to serve as features for fingerprinting web clients. This analysis complements the work by Nikiforakis et al. [23].

Cookies Cookies are the most common way to implement device identification by storing a unique identifier (usually a short string) directly on the client system. They provide an easy yet effective way of identifying and tracking systems, but do not use any of the system's characteristic features.

- *HTTP cookies* may be set and queried at the HTTP protocol level or via scripting languages such as JavaScript (unless the *HTTP-Only* flag is set). Storing cookies, however, may be disabled by the user in the browser configuration. Furthermore, HTTP cookies are only valid for a limited time span.

- *Flash cookies* use the Adobe Flash plugin to store unique identifiers at the client using so called *Local Storage Objects* (LSOs). Flash cookies are not managed by the browser's cookie policy and thus harder to remove than HTTP cookies.
- *Silverlight* and *ActiveX cookies* store identification objects directly on the client using the Silverlight, respectively, ActiveX plugin.
- *PNG cookies* consist of images that are placed in the client's browser cache, whereby the individual RGB pixel values represent the actual unique identifier. Once a PNG cookie is stored, successive requests for that image are answered with HTTP `304 Not Modified` responses, indicating that the client should load the cached version of the image, which then can be read and used for tracking.

HTTP The HTTP protocol provides several properties in the header generated at the client-side that can give clues about a system.

- The `user agent` includes the name and version string of the browser. OS version and particular system specifications might also be included.
- The `accept language` header specifies the language of the browser, while the `Accept` header field contains a list of supported mime types for the content. Browser addons and plugins (e.g., Adobe Flash) might extend this list, thus can be detected by analyzing this header field.
- The `ETag` value is mainly used for caching, however, might be misused to assign unique tracking identifiers to individual clients.
- The `X-Forwarded-For` field is inserted into a client's request header by HTTP proxies and might reveal the client's IP address.
- The `referer` header field includes the origin of the request and is only set when the user reaches the target resources, e.g., via a hyperlink.

Storage Many browsers also provide functionality to store data directly on the clients.

- The *local storage* is a client-side storage buffer that can be addressed via JavaScript and is restricted by *origin policies*, i.e., a local storage object may not be shared across distinct origins. Similar to cookies, unique identifiers can be stored on the client host.
- The *session storage* provides similar storage capabilities, however, the stored objects are destroyed when the browser application is shut down.
- *Web SQL* and *Indexed DB* are interfaces for local databases, in which objects can be stored for a longer period of time. The Web SQL interface, however, is no longer maintained, while Indexed DB is not fully supported by all browsers yet.
- The *userData* extension—for Internet Explorer only— allows storing larger fragments of data [21].
- *Caching* can be implemented by the browser object `window.name`. It is capable of storing data that is persistent over domain contexts, thus allowing cross-domain tracking. Similar to the *session storage*, the data is destroyed when closing the browser.

Browser Object Model Various browser attributes, accessible via the Browser Object Model (BOM) can be used in the generation of unique fingerprints.

- The set of available *fonts* correlates to installed applications, themes and plugins. Individual fonts might be installed by additional software.
- A list of installed *plugins* can be iterated via the `navigator.plugins` object or by probing for particular image URLs on Chrome [10]. Plugins are commonly installed to modify the browser's behavior and to add new features, but as the `navigator.plugins` object is (by default) not sorted, it might allow a more fine-grained breakdown of clients. Its order depends on the date of installation and may vary across different end hosts [23]. However, plugins such as Adobe Flash or ActiveX might also introduce further attributes that can be used for device tracking.
- Generating a *Canvas* element provides useful system information [20]. The rendering process and the pixel values of the resulting image are influenced by the rendering engine, hardware, and system-specific libraries, thus the image might vary on each client system.
- The *integer precision* may vary between different browsers and versions and thus help to distinguish client systems.
- Additional information about the browser and the system are directly accessible via the BOM. This includes: currently used *language*, the URL of the root document (*Root-URL*), current timezone, CPU and OS versions, as well as, display properties like screen width, height, and color depth.

Server-side Besides these groups of features, network specifications are also of interest for fingerprinting and are used by at least some of the examined tracking libraries. Note that the following features are gathered at the server-side and we therefore cannot make an assured statement about which library does not implement methods for obtaining these.

- The *MAC address* is the unique identifier of the underlying networking device.
- The *IP address* can also be obtained and may reveal the client's location via GeoIP lookups.
- *IP / TCP headers* might provide additional feature values for device tracking, e.g., to guess the uptime and operating system of the user's host.

Note that all HTTP-based properties might also be obtained at the server-side as well.

To investigate which features and fingerprinting techniques are used by (commercial) tracking libraries, we first collected a representative set of commonly used libraries. We analyzed popular websites using the Alexa ranking [3] and obtained a set of commonly used tracking libraries. Such JavaScript libraries leverage different features to implement device tracking and fingerprinting methods. We collected and analyzed the code of nine tracking libraries, and found that they leverage many different features. We focus on the effectiveness of these features and possible fingerprinting evasion scenarios in this paper.

In general, we observe that attributes can be easily obtained from the *Browser Object Model* (BOM) without performing sophisticated calculations. Table 1 outlines the extracted features that are used for generating fingerprints and storing unique identifiers. Additionally, we find that all libraries collect information about these characteristics: i) user agent, ii) display properties, iii) timezone setting, and iv) CPU & OS versions.

Table 1: Tracking libraries and the applied fingerprinting techniques

Properties	AFK Media	Analytics Engine	BlueCava	Device Ident	Inside Graph	Iovation	ITT	Max-Mind	Threat-Metrix
Cookies	-	HTTP	HTTP	HTTP	HTTP	HTTP	HTTP	-	-
	-	-	Flash	-	-	Flash	-	-	-
	-	-	ActiveX/ Silverlight	-	-	ActiveX/ Silverlight	ActiveX/ Silverlight	-	ActiveX/ Silverlight
	-	-	-	PNG	-	-	PNG	-	-
HTTP	Mimetypes	Mimetypes	Mimetypes	Mimetypes	-	-	Mimetypes	Mimetypes	Mimetypes
	-	-	Referrer	-	-	Referrer	Referrer	-	Referrer
	-	-	XFF	-	-	-	-	-	-
Storage	-	-	Local	Local	-	Local	Local	-	Local
	-	-	-	-	-	WebSQL	-	-	-
	-	-	userData	-	-	-	userData	-	-
	-	-	-	-	-	-	Caching	-	-
BOM	Fonts	Fonts	Fonts	-	Fonts	Fonts	Fonts	Fonts	Fonts
	-	-	Language	Language	-	Language	Language	Language	Language
	Plugins	Plugins	Plugins	Plugins	-	Plugins	Plugins	Plugins	Plugins
	-	-	Root-URL	-	Root-URL	Root-URL	Root-URL	Root-URL	Root-URL
	-	-	-	Canvas	-	-	Canvas	-	Canvas
	-	-	Integer precision	-	-	-	-	-	-

2.2 Effectiveness for Mobile Devices

Intuitively, many of the features that are implemented in the examined libraries do not work for mobile devices (e. g., Flash or Silverlight cookies). We therefore aim to study whether common fingerprint methods for desktop computers might also be applicable for mobile devices. To do so, we analyzed a set of real-world data consisting of values aggregated using common fingerprinting methods by an advertisement service provider. This data set (collected in June and July 2014) includes features collected from over 15,000 client systems. In total, 211,652 feature values were obtained from desktop computers and mobile devices. By reasons of privacy, the data was anonymized and any kind of personal identifiers were removed.

We divided the data into two subsets by filtering the user-agent string for desktop and mobile device identifiers. The first subset S_{Mobile} was extracted by filtering the complete data for mobile device specifiers. The second subset $S_{Desktop}$ consists of data of desktop computers and features the same size as S_{Mobile}. Each subset includes over 2,100 representative devices with about 35,000 feature values. These features contain many HTTP header fields and aim especially for fingerprinting desktop computers. In the following, we refer to them as *desktop feature set.*

We measured the information gain of features in each set with respect to the classes instrumenting the Kullback-Leibler divergence (KLD) [15] to obtain an information score for every feature. A higher score represents a higher entropy and hence more worth of information. The scores do not provide percentage values about detection, but allow us to compare the average information content of each feature in both subsets. The features ranked according to their information gain are shown in Table 2.

Table 2: KLD results for $S_{Desktop}$ and S_{Mobile}

$S_{Desktop}$		S_{Mobile}	
Score	Feature	Score	Feature
6.784	plugins	4.551	accept language
6.730	mimetypes	4.533	user agent
5.920	user agent	3.601	language
5.865	accept language	2.119	timezone
5.213	plugin versions	1.843	screen y
4.419	fonts	1.492	canvas
4.185	language	1.284	screen x
2.952	canvas	1.106	mimetypes
2.755	screen x	0.939	accept encoding
2.426	screen y	0.605	plugins
2.095	timezone	0.184	accept
1.750	accept encoding	0.072	color depth
1.118	accept	0.058	plugin versions
0.700	color depth	0.044	fonts

Note that almost every feature provides less information when applied to mobile devices. The fact that the scores in S_{Mobile} are generally lower compared to $S_{Desktop}$ is a first hint that features which perform well for fingerprinting desktop computers may not achieve the same precision when applied to mobile devices. The most descriptive features for desktop computers in our dataset are browser plugins and mimetypes. These attributes are standardized for most mobile devices and usually cannot be changed by the user. Due to this lack of customization ability, these two features have a low information score for mobile devices. However, the user-agent seems to provide valuable information for both desktop computers and mobile devices. The score is, though, lower for the mobile subset, meaning that a classification by the user-agent for mobile devices would be less precise than the one for desktops.

The high standardization of mobile devices results in less diversity of attributes like fonts, screen size, and color depth. In addition, there are only few possibilities to customize mobile devices: Standard browsers often do not support plugins natively, and the installation of non-standard browsers is rare.

After determining the descriptive power of features we have seen in the wild, we investigate their ability for device recognition in the following ways. For ground truth of the classification, we use a device ID which is a hash value stored in a cookie.

To analyze the two subsets from our data set, we used a J48 decision tree model. The evaluation showed that 91.45% of $S_{Desktop}$ were correctly classified using the *desktop feature set*. In contrast, the model was able to correctly classify only 37.16% of S_{Mobile} using the same feature set. The decrease in correct classification has already been foreshadowed by the information gain analysis. The low classification rate for the subset of mobile devices substantiates our claim that features which perform well for fingerprinting desktop computers are not necessarily appropriate for fingerprinting mobile devices as well.

3 Fingerprinting Mobile Devices

As shown in the previous section, existing fingerprinting techniques lack precision for mobile devices. We now propose a feature set that is particularly applicable for fingerprinting mobile devices. This feature set consists of properties and attributes that have been aggregated by instrumenting the browser environment using JavaScript. We aim to study the effectiveness of the feature set for mobile devices, even if not all of these features may be exclusively available. We divide the characteristics of a mobile device into four categories and discuss each in the following.

3.1 Browser Attributes

Browser applications already provide various information with respect to the systems' environment. We discovered that common mobile web browsers such as Android's native browser, Google Chrome, Firefox, IE mobile, Opera, Opera Mini, and Safari reveal information about the browser version, the OS, and the underlying rendering engine. Furthermore, Android's native browser, Chrome (the two most frequently used browsers on Android devices [4]), and Safari also provide the device manufacturer, model, and the browser's language. IE mobile and Opera allow the detection of device manufacturer and model as well.

Additionally, we obtain further browser attributes such as the "Do-Not-Track" (DNT) option, the capability of storing cookies, using Local Storage, and Java. We can also detect whether the browser blocks popups by default and—if newer web technologies are supported—the standard search engine.

Whereas on desktop computers features like supported mimetypes and installed plugins change with the installation or deinstallation of software, on mobile devices changes to these features are very uncommon and generally imprecise (see Section 2.2). We tried to determine if specific protocol handlers are registered with the browser (e.g., the one for Skype), thus revealing if specific applications are installed. However, this is a noisy procedure and interferes with user operations as a message will pop up asking for the application to handle this protocol. In any case, the user will be alarmed. As this is not in our interest, such features are out of scope.

To build a ground truth for evaluation, we set device IDs for newly occurring devices that are either contained in the local storage, or stored as cookie. If a device is already flagged by such an ID, the device is identified as re-visitor. We found the *File API* of *HTML5* to be too noisy for storing such IDs as the API requires user permission to store this data.

3.2 System Attributes

Due to high standardization of mobile browsers, we cannot expected to be able to distinguish devices on the basis of browser information only. For this reason, we aim to gather more information about the device system itself. However, we still use the browser in order to obtain information, and we are therefore subjected to certain restrictions due to sandboxing and limited permissions. Furthermore, we want to employ low-noise fingerprinting, i.e., we do not install any app or perform any activities that raises a user's suspicion. With these restrictions in mind, we are able to obtain the following system-wide information.

From the navigator object, the screen width and height, and the display's colordepth are extracted. Additionally, the OS name and version that are provided by the navigator are useful for our purposes. Most current versions of common browsers also yield information about the current connection type. We obtain information when the device is in a WiFi network or using a mobile connection like 3G or 4G.

Besides the connection type, we gather information about the environment of the mobile device. More specifically, we obtain the device's timezone by calculating the time offset to 13 different time points and building a hash of the differences. We also store the device's IP address and the hostname of the network node, e.g., a WiFi router. To have a more general view, the hostname is masked with a wildcard which can be used as additional feature. Hostnames often look like *ip-xxx-xxx-xxx-xxx.web.provider.com*, consisting of the device's IP address and the network provider's (sub)domain. The hostname wildcard in this example would be **.web. provider.com*, which allows a grouping of devices based on the network they are logged in. We use *MaxMind GeoIP2* [18] to determine geographical information about the current location of the device.

We also implemented an *Apple AirPlay* detector. AirPlay receivers listen for local network devices to potentially stream media content. We implemented a function that requests to stream an audio file if a mobile device is connected to WiFi and has already been identified as running iOS. This makes the AirPlay protocol return a list of available devices able to play the file. After receiving this list, we abort and withdraw the streaming request. The list of AirPlay enabled network devices may provide information about the environment (e.g., if a user owns an AppleTV).

Additional system specific attributes like active widgets, enabled/ disabled phone encryption or developer options were not accessible through any web browser. Certainly, it might be possible to check these options when running an app such as ad-trackers. However, in our scenario we are restricted to browser techniques. Additionally, we considered measuring the device's CPU and memory as well as the network-based and GPS-based location. We also developed a JavaScript-based network scanner that determines the device's local IP address next to other network devices such as routers. However, as we do not want to arouse suspicion, we decided to omit such attributes. Scanning the local network, or testing CPU and memory would result in a high load and performance loss, and determining the GPS-based location usually leads to a popup asking the user for access permission.

3.3 Hardware Attributes

The aforementioned restrictions of browser permissions lead to the fact that barely any hardware meta-data can be accessed. As such, we are not capable of obtaining identifiers like serial numbers of specific hardware elements, e.g., the camera module. Nevertheless, we aggregate the following three attributes in the browser context: the device's platform, the number of the device's touchpoints, and the availability of a vibration motor.

Additionally, we can access a device's gyroscope and accelerometers via JavaScript, which is commonly used in browser-based games. Prior work has shown that these sensors have imperfections that vary among different devices [11]. To determine these imperfections, we implemented a function to gather accelerometer and gyroscope data, and used such data as another descriptive feature. Please note that we do not have information about the users' current activities, and the device may be moving while gathering this data. To avoid distortions, we filter out these movements based on the amplitude of acceleration. Other hardware information such as the availability of a second SIM card slot or sensor specifications are not available for the browser.

3.4 Behavioral Attributes

We also implemented three functions to gather more detailed information about a device's user and her behavior. First, as we aim to learn about the user's browsing habits, we implemented a timing attack technique for history stealing [24]. The rendering of visited hyperlinks differs from the rendering of unvisited links in various browsers. This fact is used to determine whether a user has visited specific websites by measuring the rendering time of specific links using the JavaScript function `requestAnimationFrame`. In our experiments, we decided to check if a user visited the websites of Amazon, Ebay, Facebook, Google, Twitter, and Zalando—each with different top level domains—to also gain information about the user's localization. We chose these websites because we can expect them to have a large user base, and hence we can see a fair chance for a random Internet user to be logged in at one or more of them. As a limitation of this feature, every website is defined as unvisited after a user clears his browser history.

Second, we query popular websites from the user's browser for objects that are only accessible for logged-in users, e.g., a specific image. More precisely, a URL is prepared so that a logged-in user gets redirected to a specific content, whereas the website's login screen will show for a non logged-in user. This URL is called in the background. If it is loaded correctly, we can assume that the user is logged-in, whereas she is not if an error occurs. This method can be applied for several popular websites, although the URLs are slightly differently built for each site. Additionally, we load an image which is publicly accessible to detect text-based browsing. Hence, if a user disabled image loading completely, we do not classify her the same as a user who is not logged-in to any of the tested websites.

Third, we implemented a function to measure the user's typing speed. As such, a text field (e.g., a CAPTCHA) is placed on the website, which can then be monitored for user input. Once the user starts typing, a timer is triggered that stops after the user did not strike a key on the keyboard for a certain time. The average number of letters per second is then calculated and used as an attribute for user behavior fingerprinting. As the typing speed can vary for even one single user, this is not meant to be a single identifier for a person. However, in combination with the other features, the typing speed might improve our classification.

3.5 Formalizing the Approach

In summary, our feature set for fingerprinting mobile devices consists of several attributes of the device's browser, system, hardware, and a small amount of user behavior. Our aim is to develop a system that, basing on the features previously described, is able to perform two operations:

- Recognizing *new* devices, i.e., devices that have never visited our service before.
- If a device is not new, recognizing and associating it to a device that has already visited our service.

We formalize this problem as an iterative algorithm, where each iteration is related to a device connecting to the service:

For each iteration i of the algorithm, we define a set of known devices:

$$K^i = \{k_1, k_2 ... k_n\}, n, i \in \mathbb{N}$$

where n is the number of devices that are already known by the system at the current iteration step, and i is the *iteration index.*

Then, we define a set of feature vectors:

$$F^i = \{A_1^i, A_2^i ... A_n^i\}, n, i \in \mathbb{N}$$

where A_n is a generic *set* containing the *feature vectors* associated to the accesses made by the *generic* device k_n at the current iteration.

This can be expressed by:

$$A_n^i = \{f_{n1}, f_{n2}, ...f_{na}\}, a, i, n \in \mathbb{N}$$

where a is the *number of accesses* made by the device k_n at the current iteration.

The generic feature vector f_{na} is then defined by:

$$f_{na} = \{m_{na1}, m_{na2}...m_{nad}\}, n, a, d \in \mathbb{N}$$

where d is the number of features we described above.

In this formulation, the following is also valid, i. e., each device k is associated to one feature vector set A:

$$A = K^i \to F^i$$

For each device $k_u \to A_u$ that visits our service, we initially suppose this condition: $A_u = \{f_u\}$, where f_u is the generic feature vector associated to an input device. Under this, we have to find the *known* feature vector $f_{min} \in A^i_{min}$ that belongs to the known device k_{min}, and that is *most similar* to f_u. This vector is given by this formulation:

$$f_{min} = \underset{f \in A^i, A^i \in F^i}{\arg\min} D(f, f_u)$$

where D is a *dissimilarity* function among feature vectors, i.e., a function that measures how much two feature vectors differ to each other.

Furthermore, let δ be a *dissimilarity threshold*, we define:

$$K^i \cap k_u = \begin{cases} k_{min} & : D(f_{min}, f_u) \leq \delta \\ \emptyset & : D(f_{min}, f_u) > \delta \end{cases}$$

The first condition means that if the dissimilarity between the feature vector f_u and f_{min} is lower than the threshold δ, then k_u is already *known* by the system and corresponds to the device $k_{min} \to A^i_{min}$. This also means that the actual set of devices does not change in the next iteration, thus obtaining this: $K^{i+1} = K^i$. The corresponding set of feature vectors A^i_{min} must be updated with the latest, recognized, access:

$$A^{i+1}_{min} = A^i_{min} \cup \{f_u\} \quad \text{and} \quad F^{i+1} = F^i \cup \{A^{i+1}_{min}\}$$

The second step is *recognizing* the device k_u, depending on the results of step 1. In particular, the second condition means that if the dissimilarity between the feature vector f_u and f_{min} is higher than the threshold δ, then $k_u \to A_u$ is defined as *unknown*.

Because of that, $k_u = k_{n+1}$, as it is a completely new device that must be added to the known devices list, and therefore we obtain a new set of devices $K^{i+1} = K^i \cup \{k_{n+1}\}$. Consequently, we define a new set of *known* feature vectors for the next iteration: $F^{i+1} = F^i \cup \{A_{n+1}\}$.

The iteration index i is increased by one so that the system will be ready for the next iteration.

4 Evaluation

We implemented the features presented in the previous section and built a fingerprinting system based on the proposed approach. To gather real-world data, we set up a publicly available online survey that can be visited by any Internet user to check whether her mobile device can be tracked using our fingerprinting methods. Most features (esp. the browser attributes) are gathered via GET and POST requests, while we leverage callbacks for asynchronous features. More precisely, we created a PHP parser to catch the return values of these callbacks. This is necessary for accessing the browsing history, fingerprinting the accelerometer, and measuring the typing speed. Gathering this information takes more time than querying navigator objects. The measurement of typing speed has been obtained by including a text field where the user has to type two words that are shown in a CAPTCHA. Figure 1 shows a screenshot of our online test.

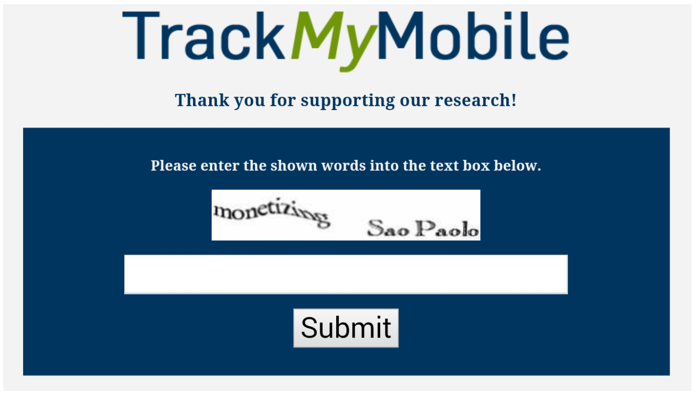
TrackMyMobile
Thank you for supporting our research!
Please enter the shown words into the text box below.
monetizing
Sao Paolo
Submit

Figure 1: Screenshot showing CAPTCHA that is used to measure typing speed

The CAPTCHA for measuring typing speed is the only interactive element on our website. Hence, if a user was not aware of her system getting fingerprinted and if the CAPTCHA was not included, she would not be able to notice the fingerprinting process without the help of additional software.

We carefully addressed ethical considerations of experimentation and data collection. So, we conducted our experiments such that:

1. all participants were informed about the nature of the experiment and how their data would be used,

2. all participants had the opportunity to *opt-out* at any time during data collection,

3. all data was stored using non-personal information to protect the privacy of participants and

4. all participants were allowed to view and received feedback on the data provided as an incentive for their participation.

We did not collect any personal identifiable information and all participants remained anonymous.

4.1 Data Set

For the following experiments, the data was assembled by our online survey described above. We spread the link to our online service via three different mailing lists, addressing university students, IT security researchers, and persons without any IT expertise. In total, almost 900 users participated in this study of which 724 were using a mobile device like smartphone or tablet. Of these mobile users, 459 accessed the test more than once over a duration of four months. These re-visitors who participated at least twice are recognized by a cookie ID and a local storage ID that serve as a ground truth for our evaluation. This choice is aimed to correctly evaluate the capabilities of the system to recognize devices that have visited our service, and to detect previously unseen devices.

In total, 45 features of the previously explained categories browser, system, hardware and behaviour were collected. These features used for fingerprinting are of different data types including integers, floats, hashes, bits, strings and plain texts. An exemplary list of these features, their data types, and a real-world value is presented in Table 3.

As first data operation, we performed an encoding for non-numerical features. By this procedure, a number will be assigned to every unique value for all features so that every feature occurrence can be represented by a numerical value enabling fast comparisons in further analyses. Note that the identifier used as ground truth (the ID stored in a cookie and the local storage) contains random characters as well as a time-based component to avoid collisions of identifiers. There may be an information overlap between single features, e. g., `operation system` and `is _Android`, which seems redundant but enables swift analyses and hence faster results and insights. During the machine learning process, these redundancies are eliminated.

To have a ground truth, we only take revisitors into account because our goal is to show the capabilities of the selected features at recognizing known devices without relying on cookies or other unique identifier. The recognition of visits from known devices is carried out by resorting to the "nearest neighbor" approach. To uniquely identify each device, we computed a hash value from the feature values associated to each device. Furthermore, we encoded every occurred value of all features that are naturally not numerical to accelerate comparison operations.

As we did not collect personal information about the participants of our experiments, we are not able to provide information like their age, technophilia, or any demographic characteristics. Although we employed in our work a comprehensive feature set, future technologies might lead to features that can also be used to fingerprint mobile devices.

4.2 Feature Distribution

Device fingerprinting will benefit from features with high diversity among the devices, whereas features with a small distribution of values are not meaningful for recognizing devices. We now provide insight into the distribution of the features by commenting on the most representative ones.

The distribution of feature values allows a first grouping of devices we have seen. But the number of distinct values of a feature does not necessarily allow to judge their importance. In general, a feature with many distinct values like

Table 3: Feature Data Types and Example Values

Feature	**Type**	**Example**
devicefingerprint	string	4812169833755445458
revisitor	bit	1
ismobile	bit	1
cookie_id	string	QoSQIymCwjg0augzs D41-1415043670.767
localstorage_id	string	rQG4fVJaDBNFtOyKd CL1-1415011415.67
mimetypes	text	[{”n”:”video/3gpp2”,”d”:”3GPP2 media”,”f”:”3g2,3gp2”}, ...]
mimetype_hash	hash	b96eebf2fd3fff0e165d77e75474ffaf
plugins	text	[{”n”:”Shockwave Flash”,”d”: ”Shockwave Flash 11.1 r115”,...}]
plugins_hash	hash	4e23a836cea77cf4af09affff2b64a75
plugins_num	int	6
canvas_hash	hash	ea907f4cd06cf0f310d7acf62f2ffff6
useragent	text	Mozilla/5.0 (...)
vendor	text	Google Inc.
productsub	text	20030107
is_chrome	bit	0
popup_blocker	bit	1
navigatorlanguage	text	en-en
filesystem_access	bit	1
cookies_enabled	bit	1
dnt_enabled	bit	0
java_enabled	bit	0
loginstatus	bitstring	10111
history	bitstring	1000100000
screen_height	int	960
screen_width	int	600
display_colordepth	int	32
display_orientation	text	landscape
platform	text	Linux armv7l
operation_system	text	iOS 7.1
is_Android	bit	0
is_iOS	bit	1
touchpoints	int	5
has_vibration	bit	1
airplay_ref	bit	1
typingspeed	int	229
accelerometer key	float	938.143359751
connection	text	wifi
hostname	text	ip-xx-xx-xx-xxx.web.provider.com
hostname_wildcard	text	*.web.provider.com
timezone_id	int	662525310
ipaddress	string	xx.xxx.xx.xxx
country	text	Germany
city	text	Bochum

accelerometer benchmarks or user agent may be good to distinguish devices. Though, if there were only few devices with the “Do-Not-Track” option enabled, these could be discovered well although this feature only allows two distinct values.

We are able to make statements about the share of specific features which gives us information about the devices in our set. Exemplary, we discuss the occurrence of ascertained values for the following four features:

1. Operating System
2. Platform
3. Vendor
4. Navigator Language

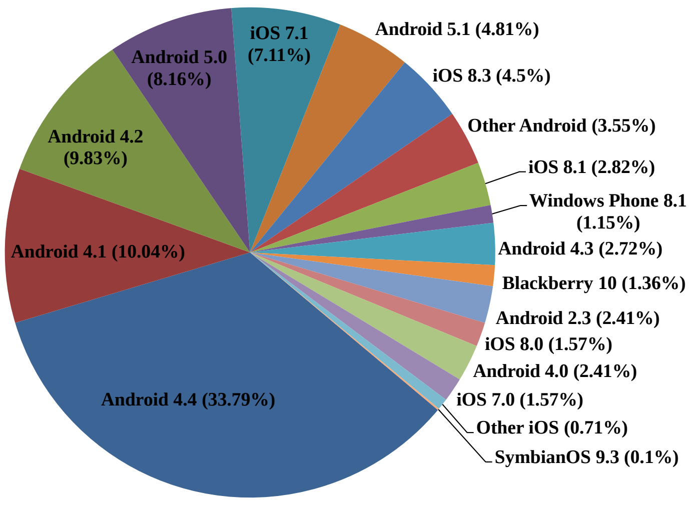

Figure 2: Distribution of Operating Systems

Figure 2 illustrates the distribution of operating systems in our data set. The majority of devices (about 78.79 %) are Android (mostly version 4.4) systems, followed by iPhones with about 15.56 % (mostly iOS 7.1). Alternative operating systems for mobile devices like Windows and Blackberry are almost negligible with about 1.15 % and 1.36 %. These shares in our dataset represent the actual market with Android having 78.9 % and iOS having 17.9 % market share while Windows has 2.5 % and Blackberry as well as other operting systems have 0.4 % at the time of this work [14]. Consequently, the most represented browser is Chrome used by 54.64 % of the participants.

As most Android devices operate on ARM, this platform type is the biggest share in our data set. Figure 3 shows the distribution of different platform types. We see that ARMv7 is the most common with about 75 %, followed by iPhone with about 17 %.

As most devices that took part in our test were running Android, also the most represented vendor is Google Inc., making about 74 % of the devices. Again, the second largest share of about 18 % is held by Apple Computer Inc. while UCWEB Inc., Yandex N.V., Research In Motion Ltd. as well as Opera Software ASA are almost negligible like shown in Figure 4.

In contrast to these features having a clear majority, the navigator language is more diverse. Although most participants of our online test have their language set to German or Italian, even for those two languages there are different feature values: German includes `de`, `de-de` and `de-DE` and other languages have these peculiarities as well. Hence, there are more groups of devices which can be distinguished by this feature. As Figure 5 shows, many small groups exist, making the navigator language a valuable piece of information for fingerprinting. Additionally, this feature allows to draw conclusions about a user's home country.

Additionally to these features, we collect behavioral data of the device's user. Especially logged-in accounts and browser history turned out to be of interest

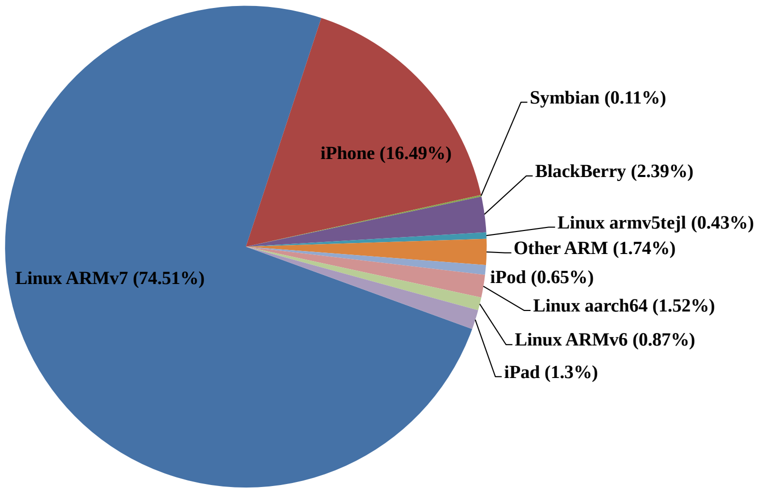

Figure 3: Distribution of Platform

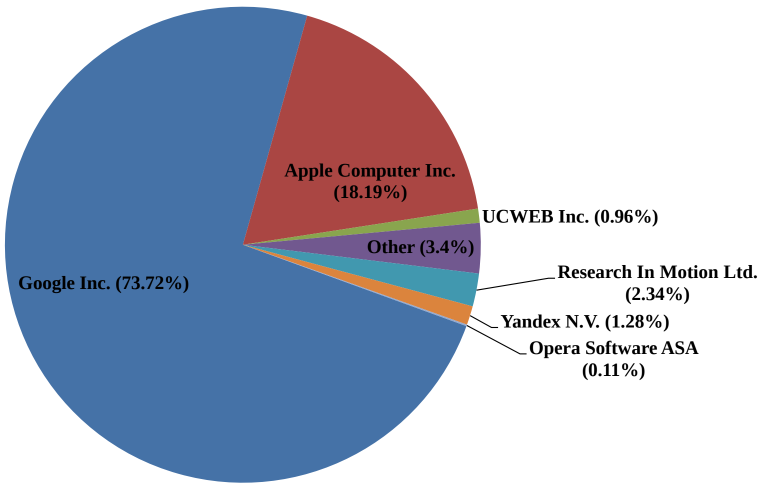

Figure 4: Distribution of Vendor

as there are wider distributions of feature values. Although 51.4,% of the users which participated in our test are not logged in at any website and also 37.21,% users have not visited those websites we checked there is sufficient diversity in these features. Especially for browsing history many small groups with different values exist, which is advantageous for fingerprinting. In combination these information give an impression of the user and her device. By assembling intersections of features even those with few different values become a valuable discriminator for recognition.

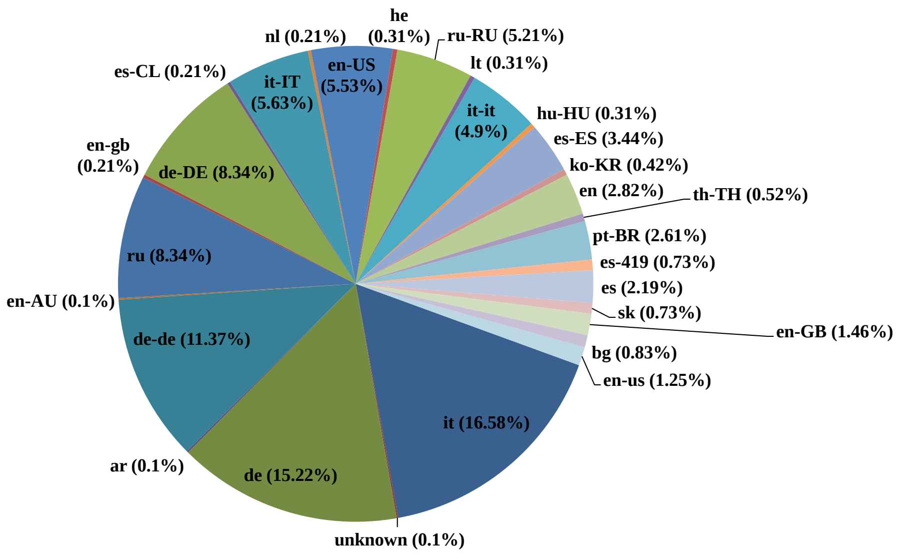

Figure 5: Distribution of Navigator Language

Furthermore, we found 96.25 % of the devices allowing access to their local file systems, 88.22 % blocking pop-up websites but only 6.36 % having enabled Do-Not-Track. In combination with other feature values, especially the smaller shares of these features may be valuable information for device recognition.

The number of distinct values of a feature is not necessarily related to their importance for device recognition. For instance, if a few devices had the "Do-Not-Track" option enabled, such devices could be well grouped, even if this feature only allows two distinct values. Nevertheless, we find features with many distinct values such as accelerometer benchmarks or user agent to be of high relevance to distinguish devices.

We define a feature as *volatile* if, even for the same device, such feature could be characterized by multiple values. This volatility can occur for attributes that change due to *environment*, *events*, or *actions*. For example, the accelerometer data provides very precise float values of a devices' sensors, which results in slightly different values for the same device. Different *environments*—a user may be in an accelerating bus once and sit still in a room another time—may cause this difference of data. That is why such values need to be grouped, or in some way condensed, to be a useful feature.

We found accelerometer data—represented by two benchmark features—to be highly distributed giving an almost unique value for every device taking our test. Typing speed and browsing history have many different values like we expected, because these are behavioral features which are very discriminative for a device's user. Although, there are not so many distinct values like for e.,g. accelerometer data because these features stay the same even for revisitors of our online test. Hence, while accelerometer benchmarks may vary even for the same device, the browsing history of a person stays the same until it is deleted manually. In contrast, we expected the login status to have more variety for the same reason. But as there are many apps available (e.g., for Facebook) it is not likely that users are logged in at Facebook with their mobile web browser. A few features are binary, naturally allowing only two different values: Cookies Enabled, Java Enabled, Display Orientation, Filesystem Access Allowed, Vibration Ability, Popup Blocker Active, DNT enabled, browser is Chrome.

However, device recognition becomes easier, as more data about it is obtained. The hostname attribute exemplifies this situation: If a user re-visited our experiment with two different connections, e.g., wifi and mobile network, both network node hostnames will be registered with this device. Recognizing the device when using one of these known hostnames will be easier. Changing the mobile cell is an *event* that also affects this feature. Furthermore, the devices' network hostname is often used as a native identifier for network providers. Hence, we are able to divide the mass of web users into groups based on their network-based location or at least their ISP. Note that volatile features have to be treated carefully. While some features need to be condensed reasonably (like the accelerometer data), other features provide more and more information with every change (like the hostname).

The number of IDs stored in cookies and local storage is also higher than the number of re-visitors. This can be explained by the *action* of deleting cookies. If a user deleted cookies or local storage after taking our test, and then revisits the website, a new ID will be generated and set. The fact that there are less cookies than overall participants indicates that not everybody deleted their cookies and local storage afterwards.

We found typing speed and browsing history to have different values per user by trend, which makes them very discriminative for a device's user as they stay the same for the revisitors of our online test, even when cookies and local storage are cleared. Whereas accelerometer benchmarks may vary even for the same device, the browsing history of a person stays the same until it is deleted manually. We expected the login status to act similar for the same reason, but it turns out that only a minority of users use the browser to login to services which provide an alternative app. Users tend to use specialized apps (e. g. for Facebook or Twitter) instead of their mobile browser for these services.

Ultimately, device fingerprinting will benefit from features with high diversity among the devices while features with a small distribution of values in their own space will probably not be meaningful for recognizing devices.

4.3 Recognition of Mobile Devices

In this section, we describe our implementation of a system for recognizing mobile devices realizing the formalism and features described in Section 3. We also present the experiments conducted to assess the system's performance and the dataset.

Implementation. To recognize mobile devices by using the features described in Section 3, we developed a system that complies with the following properties: First, the characteristics described in Section 3 are used as features, which will be properly preprocessed to perform an easier computation of the dissimilarity function. Second, the system applies a nearest-neighbor matching approach (essentially a 1-NN) to perform the detection of known and unknown devices. This choice is related to the matching-nature of the problem, for which this classifier exhibits good performances. More precisely, our approach resorts to the dissimilarity function D in order to extract the closest points to the input feature vector f_u in the feature space. We define D among two feature vectors f_1 and f_2 in this way:

$$D(f_1, f_2) = (w \cdot c(f_1, f_2)) \,/\, \sum_{i=0}^{d} w_i$$

with d as number of features, and $w = \{w_1, w_2, ...w_d\}$ represents the feature weight vector that is calculated by means of its information gain IG. Thus, for each m_i-feature, we calculate its weight as follows: $w_i = IG_i = H(D) - H(D|m_i)$, where $H(D)$ and $H(D|m_i)$ are the entropy values for a specific device (considering all its accesses) before and after observing the m_i-feature. We also define $c(f_1, f_2) = \{c_1, c_2, ...c_d\}$ as a vector whose generic component c_i is calculated as follows:

$$c_i = \begin{cases} 0 & : f_{1i} = f_{2i} \\ 1 & : Otherwise \end{cases}$$

As all the features in f_1 and f_2 are encoded as numbers, they contribute to the distance only if they have *different* values. As described in Section 3.5, the system determines the feature vector f_{min} with the lowest distance D from f_u. If $D(f_u, f_{min}) < \delta$, the devices described by f_u and f_{min} will be matched. Otherwise, the input feature vector f_u will be associated to a new device and added to the system database.

Experimental Settings. We performed an experiment in which our system was designed to detect *unknown* devices, and to match at the same time *known* devices to the correct ones. We run such experiment under two possible scenarios:

1. Single-Iteration mode: In this scenario, we suppose the website has already been visited by a number of devices. The goal is recognizing if new devices have visited the system *without updating* its list of known devices. This is done to verify how many new devices could be correctly detected by the system during a single iteration. At the same time, the system must also be able of recognizing multiple accesses of the same device.

2. Multiple Iteration mode: In this scenario, the visits of each device are considered one after the other, and they are simulated at different times. After each iteration, the list of known devices is *updated* by adding the features related to the new visit. This procedure completely reproduces the algorithm that we described in the previous section.

We believe that these two scenarios are representative of typical real-world situations, and can give a good overview of the general performances of our system.

4.4 Single-Iteration Experiment

In the first experiment, we evaluated the matching properties of our system when a database of known devices that visited the system is built beforehand. For each device, the features related to different visits are stored. The aims of this experiment are the following.

First, detecting *known* devices, i.e., finding in the database the device that correctly associates to the input of our system. We represent this case with the term *match*. Second, detecting *mismatchings*, i.e., successfully performing two operations: **a)** correctly distinguishing a never-seen device from all the ones included in the database; **b)** correctly recognizing all the devices in the database that are different to the input of our system. We represent this case with the term *reject*. The choice of the terms *match* and *reject* comes from the similarity of this problem to the ones found in biometrics. In a typical biometrics setting, the system should be able to authenticate (*match*) or refuse (*reject*) the user that tries to access to its system. We believe that such terminology can be useful in the scenario at hand.

In this first experiment, we split our dataset into three pairs. Each pair is composed by a reference set and a test set, respectively composed by 206 accesses. Using multiple pairs of reference and test sets reduces the possibilities that specific performances are obtained because of a lucky/unlucky reference-test division. We then extracted the feature weights with a ten-fold cross validation calculated on the reference set. Finally, given each reference-test pair, we verified the performances of our system on the corresponding test set. Such assessment has been repeated by considering different scenarios: **a)** all features are used for the detection; **b)** the most discriminant features (i.e., features with the highest weights) are progressively removed from the feature set. This evaluation is of interest, in particular when important information such as cookies or IP address is removed. Figure 6 shows the ROC (Receiver Operating Characteristic) plot that measures under multiple scenarios the *average* performances of the system on the three reference-test splits. On the y axis we report the amount of correctly matched devices (*true positives*), whilst on the x axis we report the errors in rejecting devices (*false positives*). Each point of the ROC curve corresponds to a value of the threshold δ, and the optimal threshold is given by the point that is closer to the upper-left corner of the plot.

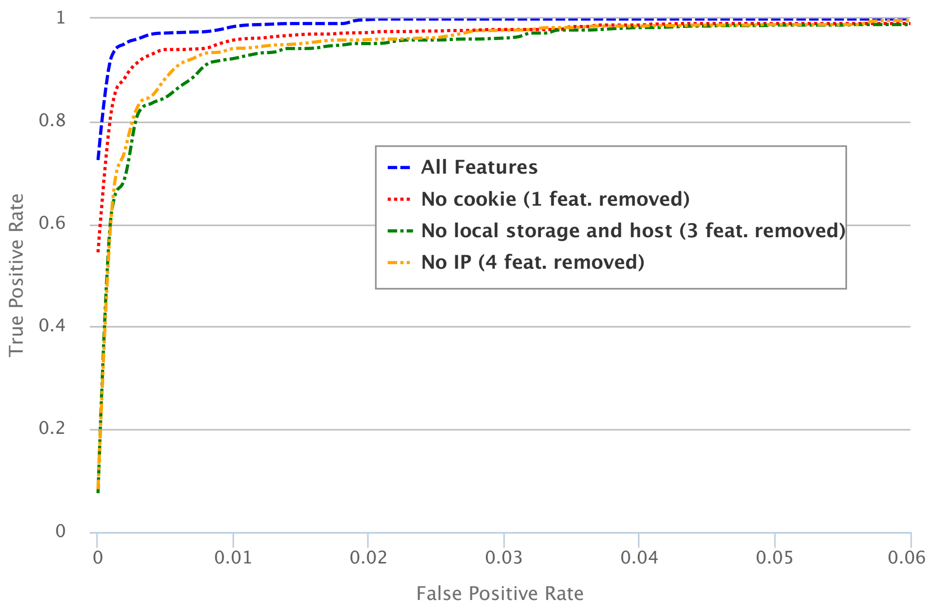

Figure 6: Average ROC performance chart for the single-iteration experiment.

From the obtained plot, we observe that the system has a very good performance at detecting new devices and at recognizing already seen ones. Performances do not change much when features like cookies, hostnames or IP addresses are excluded from the feature set. This is because weights are distributed on the features in a way that no feature is completely dominant on each other. For the same reason, when all features are considered (including cookies), the system does not have 100% detection rate with zero false positives. Of course, this figure would change by increasing the weight assigned to cookies. However, this would compromise the general performances when such dominant features are not considered. It also seems that excluding the IP address in addition to beforehand omitted features may increase the overall recognition rate and lead to a better performance. This difference is due to the way, our k-NN classifier handles the features' weights: When leaving out any feature, the algorithm re-assigns weights so that features which were not particularly important may get a higher weight and hence be more essential in the new scenario. This also affects features with a beforehand high weight as they become less important in relation during this procedure. However, the difference between the recognition rates are not significant when taking the variance into account. Although k-NN re-weights all features in every scenario, the differences between all examined cases are marginal.

4.5 Multi-Iteration Experiment

The aims of this experiment are the same of the previous one, but in this case we assess the performances of the system by strictly following the algorithm that we proposed in Section 3. We therefore simulate that all the devices in the database visit our service *one after the other*. The order of the visits is strictly random. At the first iteration, the reference set contains just one sample, and it will dynamically increase its size after each visit. In this scenario, the system has no supervised knowledge, and will progressively adapt itself to recognize new devices. Of course, this means that the system might exhibit more matching errors, especially when the size of the reference set is particularly small. Figure 7 shows the ROC curve by considering the same scenarios as those of the Single-Iteration Experiment. As the reference set dynamically increases after each visit, every score used to compute the ROC has been calculated on different reference sets.

From the attained results, we observe that the performance of our system is significantly worse than the previous experiment. However, this was predictable as this experiment starts with only one sample. Errors in matching known devices and failures in recognizing new devices will affect the performances. However, even in these conditions the system provides good performances with a decent number of false positives.

In this case, we also observe that the dependency of the performances on the most discriminant features is much more evident. In fact, without considering the IP address or the local storage ID, system performances exhibit a drastic decrease. We speculate that highly discriminant features are important for a more accurate matching with few reference samples. Figure 7 also confirms the trend shown in Figure 6: devices can be tracked, to a certain extent, even without resorting to cookies.

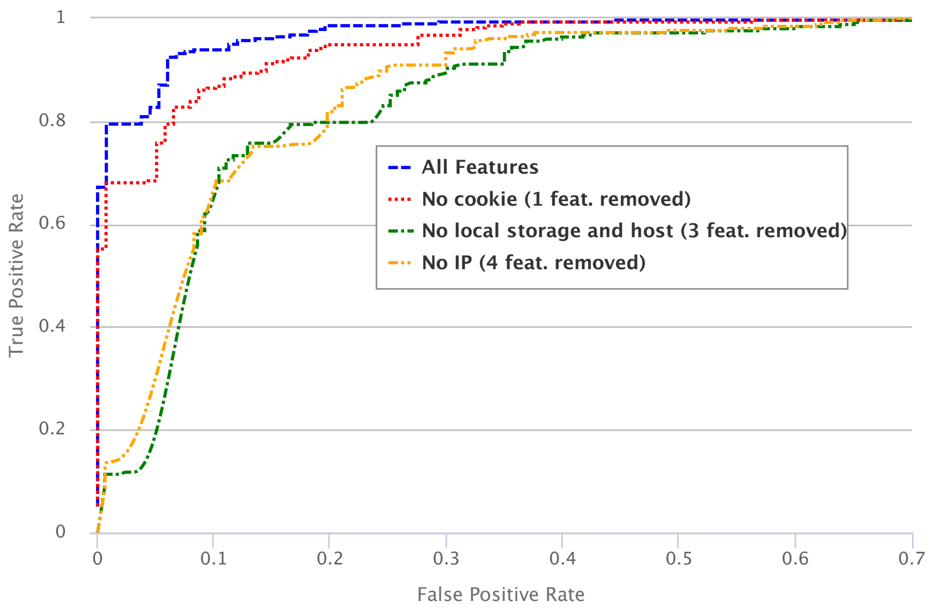

Figure 7: Average ROC performance chart for the multi-iteration experiment.

Note that our detection approach might suffer from slowdowns if the database was filled with millions of accesses due to the nature of the used machine learning algorithm. However, in such scenario it is possible to apply techniques like prototype selection to reduce the size of the database without loosing accuracy.

5 Analysis of Evasion Resistance

A crucial aspect of our evaluation is the robustness of our system against *evasion attacks*. In our setting, an evasion attack can be defined as an attempt by a *user* to avoid her device being recognized by a service she visits more than once. The actions carried out by the user to evade detection affect elements such as system applications (e.g., browsers) or connection properties, and features values related to such elements will be accordingly changed.

5.1 Changeability of Features

To shed light on the robustness of the proposed fingerprinting technique, we analyze the degree of changeability of the employed features. Whereas it might not be challenging for an experienced user to change some of these features, other features cannot be easily changed. For example, there are features that change according to the context (e. g., the timezone changes when the user is traveling), but there are also immutable ones (e. g. hardware attributes).

Among the set of features whose values can be directly changed by the user, we include browser-based features and other features related to user's *behavior*. The following list includes all the features that the user can change with different degrees of difficulty:

- popup blocker active
- DNT option enabled
- Java availability
- cookies enabled
- autofill forms
- filesystem access
- cookie ID
- local storage ID
- login status
- browsing history
- navigator language
- user agent
- display orientation
- AirPlay availability
- typing speed
- browser plugins
- accelerometer data

There are several easy-to-switch binary features. For example, the following elements contained in the set are linked to user options: *Java, popup blocking, "Do-Not-Track", autofill forms, filesystem access, cookies,* and *local storage.* Although the user can block cookies and local storage usage for websites, this may result in limiting the functionality of the visited websites. For example, most online shops use cookies and/or local storage to track the items visualized by the user during her visits and to store items in an online "shopping cart". Thus, a user would probably not want to completely disable this functionality, but might regularly delete cookies and local storage data.

Additionally, the user can influence the features related to the login status and history by regularly logging out from websites that require authentication and regularly clearing the browser history. If an advanced user is aware that these features are gathered for fingerprinting purposes, she could easily create fake accounts to alternately log in and out, in order to induce as much randomness as possible. The same applies to the history feature.

To alter the navigator's appearance, a user could easily set a different language. A user is also free to use a different browser that changes the navigator user agent completely. This would also affect the canvas fingerprinting method. However, this action requires the installation of new software and not just changing a browser's configuration. On the other hand, if the reference set for a given device contains samples related to visits by different browsers, a browser change will not affect the matching performances.

Some browser-independent attributes can also be influenced by the user: For example, display orientation can be easily changed, but as orientation may affect the usability of certain websites, it cannot be changed arbitrarily. The features related to the availability of Apple AirPlay can be influenced by deactivating the streaming service by default and enabling it when used. Anyway, heavily resorting to this procedure can affect the usability of the system. Typing speed and accelerometer measurements can be tampered with, too. If a user is aware that the typing speed is measured, it is easy to manually add randomness (e.g., by typing slower than usual). To distort accelerometer data, the user could unpredictably move the device the whole time. Besides the user's direct sphere of influence, the following features are likely to change on certain events:

- IP address
- hostname
- timezone
- country
- city
- connection type

With the exception of the connection type, these features depend on the location of the device. A mobile device's IP address is most likely to be volatile, as it changes according to network providers' rules. The hostname and its corresponding wildcard are based on the IP address. Timezone, country, and city are determined with the help of the device's IP. Consequently, if a user spoofs the device's location, these features will change. Using a proxy (e. g., an anonymization service) invalidates all location-based attributes. The connection type might be changed by the user as well, for example by alternatively using public WiFi services, and mobile data connections.

Whereas some features can be directly influenced by the user, or can be changed according to the environment, others are (almost) completely immutable for a user in a normal setting:

- platform
- operating system
- vendor
- productsub
- vibration availability
- screen height
- screen width
- display color depth
- no. of touchpoints
- is iOS / is Android

These features depend on the device's hardware or operating system, and changing them would require a much higher effort. Even if a user is able to install another operating system on a device, the platform, vendor, and productsub cannot be changed. It may be possible to feign other vendors or use alternative display resolutions; however, faking the availability of vibration, number of touchpoints, or display data needs a higher effort. We therefore expect these features to be immutable in our scenarios.

Our assumptions are that first the user has to *know* that device attributes are captured. Additionally, she needs to know which specific attributes are gathered and used for fingerprinting. If a user is aware of this information, it would be possible to do a purposeful and selective randomization/faking of the attributes. If a user is only aware of the fact that the device gets fingerprinted but does not know about specific attributes, it would still be possible to perform a general randomization of common features to deceive standard fingerprinting techniques.

5.2 Evasion Attacks

The overall goal of evasion attacks is to prevent the fingerprinting system from recognizing a mobile device by changing specific properties of the device so that its feature values will be accordingly changed. However, this is not an easy task for the user because the probability of performing a successful attack also depends on the *knowledge* that the user has of the fingerprinting system. This means that a user that wants to evade the system should have knowledge of several elements:

- all the features that the system uses;
- the system detection algorithm and (in our case) its measure of dissimilarity between devices;
- previous accesses that have been made to the system, and their impact on the feature set. This is crucial, as if some changes could create a greater distance among one access and the other, they can reduce the distance with another access (from the same device) made with different resources;
- the system decision boundary, which depends on the nearest-sample matching rule. This is particularly critical when the same device accesses the website multiple times.

Collecting the knowledge listed in the previous points implies that a user has *perfect knowledge* of the system [6]. Obtaining such information for an outsider is a difficult task, and might not be feasible most of the times.

For this reason, the reported evaluation of the system assumes that the user has *limited knowledge* of the system. This means that she knows the properties of some devices that accessed the website (e.g., their browser or their proxy settings). Her *goal* is changing the parameters of her own device so that they match, as much as possible, the ones of another device. The *rationale* behind this attack is that the access with the modified parameters will confuse the system and will stop it from correctly detecting the device (as it should reduce the dissimilarity measure from other devices). It is worth noting that the user does not know the impact that her actions will have on the features.

To provide events that can be concretely realized by the user with relatively low effort, we imagine four scenarios under which the user makes changes. These scenarios can be manually achieved by changing a device's configuration, or can be automatically obtained by using specific applications.

1. **Second browser** Users can create variance within the feature set by installing a second browser and alternately using two browsers. This would affect the following features: *(i) user agent, (ii) canvas hash, (iii) plugins.*

2. **Second browser with different settings** In addition to alternate between two browsers, users can adjust the settings of one browser in contrast to the settings of the other browser, e. g., enabling DNT for one browser and disabling it for the second. Hence, several features that are extracted from these settings would change, creating more differences between the two browsers. For example, this could be achieved by deleting cookies and local storage after every usage; by changing the navigator language; by using popup blocker and DNT-header; by logging out from websites and clearing the browsing history everytime. These actions would change the first scenario's features and would additionally modify these: *(i) local storage ID, (ii) cookie ID, (iii) navigator language, (iv) popup blocker active, (v) DNT option enabled, (vi) login status, (vii) browsing history.*

3. **Proxy** Besides changing settings related to the device directly, users can influence features used for fingerprinting by using a proxy connection. This could be done by resorting to manual configurations, or by employing a proxy application. Such a behavior would change a client's location-related features: *(i) IP address, (ii) country, (iii) city, (iv) hostname, (v) hostname wildcard.*

4. **Two browsers and a proxy** The combination of the preceding actions that can be taken by users builds the strongest scenario. Using a second browser with differently adjusted settings, and resorting at the same time to a proxy connection affects all of the above listed features.

5.3 Evasion Results

For this experiment, we considered the *whole dataset* as a training set. As a test set, we changed the features of each sample of the training set, based on the scenarios described above. As target values, we randomly selected values that belonged to other devices. This has been done to guarantee that the values were coherent and not just randomly chosen. We repeated this experiment ten times for each scenario by always using different target samples. It is worth noting that we tried to simulate the scenarios in the most accurate way. For example, we considered the fact that a device using Android could not switch its browser to one belonging to IOS. For instance, it was not possible to change Chrome to Safari in a non-IOS device.

Figure 8 shows the average ROC curves for our system under the aforementioned evasion scenarios. Since some ROC curves are really similar to each other, we also computed the portion of the Area Under the ROC curve (pAUC) for values of false positives between 0% and 1%, and for the ones between 0% and 10%. From these results, we can observe the following facts:

a) Simply changing the browser or using a proxy does not impact the system performances. Presumably, this happens because the user is not aware of the devices that the system has already seen. Changing the browser might be risky, as the system might be more sensitive to the new browser than to the previous one. Furthermore, the distance function that we chose also depends on the *number* of features that are different between the two samples. The more features the user manages to change, the better the attack will be. In this case, only changing browser or proxy brings too few features changes, degrading the effectiveness of the attack.

b) What really impacts the performances of the system is changing the browser *and* its settings. Although the detection rate does not completely break, we notice a significant drop of around 60% at zero false positives. We assume that this is due to an increased number of feature changes. This is inline with the action taken by the user: by completely changing the browser settings, she significantly affects the fingerprinting capabilities of the system.

c) When the user changes her browser settings and uses a proxy, we observe a *complete crash* of the detection rate with zero false positives. To make the system provide good detection values, false positives have to increase up to 10% for a detection rate of 70%.

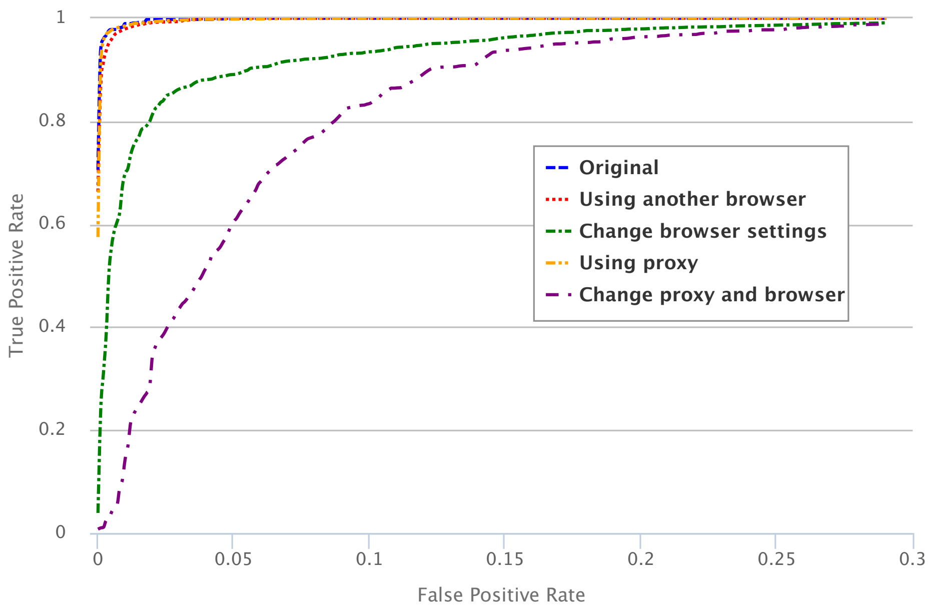

Figure 8: Average ROC performance chart for our system under multiple scenarios of evasion attacks.

The results of this experiment suggest that when a user changes the browser settings and resorts to a proxy, she is able to completely evade a system that does not make any mistakes at detecting devices that have never been seen before. This behavior creates serious concerns, as errors at detecting never-seen devices might compromise the functionality of the system in the long term.

To conclude, this experiment shows that evading mobile device fingerprints is possible, but not so easy as it might be expected at a first glance. The user has to produce a significant effort to obtain an effective evasion. At the same time, our analysis results show that even a complete evasion is possible when the user resorts to a second browser with modified settings, and to a proxy.

6 Related Work

Several studies showed that client fingerprinting is a valuable method and that it is used in practice for user tracking, fraud detection, or advertising [12,16,20,23]. Fingerprinting does not aim to replace stateful fingerprinting via cookies, yet. However, fingerprinting is harder to detect and allows user recognition of those who deleted their browser's cookies [2].

A detailed investigation of how web browser fingerprinting works has been conducted by Nikiforakis et al., showing that user privacy could be compromised by modern fingerprinting techniques [23]. The authors' results are mostly limited to full-weight browsers installed on desktop computers which are far more customizable than browsers on mobile devices. As the habit of mobile browsing increases constantly, we reviewed browser fingerprinting techniques in the context of mobile devices.

A first study on mobile web tracking was performed by Eubank et al. [13]. While stating that mobile tracking is an under-researched area, the authors show differences between desktop and mobile fingerprinting, focussing on cookies and HTTP headers. However, this work is limited to Firefox Mobile on Android, which is not the most common browser on mobile devices. Our study contains data from different browsers as well as different systems and devices, and it is based on a larger feature set.

A study of canvas fingerprinting and evercookies showed that users have only little to oppose against these techniques [1]. The proposed mitigation is to ask the user for permission at every read event which is, as of yet, only implemented in the Tor Browser [9]. Evercookies are hard to defend on desktop computers as common browsers do not provide an interface for checking and deleting the local storage or Indexed DB, and Flash storage is not isolated. We found that the user cannot browse the local storage or IndexedDB on mobile devices, too. Newer versions of Apple's iOS (7 and later) combine the functions of deleting cookies and the local storage. However, we see that many users do not delete their cookies and data regularly. Even worse, if a user's device can be recognized without local storage data and cookies, the information about if and how often a user clears cookies and website data may become a new feature describing the user's behavior and awareness.

An approach to defend against fingerprinting is the randomization of characteristic attributes. Nikiforakis and Livshits have shown that this technique can be used to deceive especially the fingerprinting of installed fonts and browser plugins [22]. *Firegloves* is a proof-of-concept Firefox plugin following this approach [7]. The drawback of randomization is its noisiness: If a feature is randomized on every access, sophisticated fingerprinting techniques could repeatedly perform measurements to determine the randomness and finally obtain the unrandomized features. Also, randomizing the lists of fonts and plugins cannot mitigate fingerprinting mobiles. Furthermore, adding randomness on browser or system features may cause unwanted effects on websites as it is hard to distinguish between a legitime functionality and fingerprinting. If random values were added to the screen's width and height attribute for instance, a website might display elements in wrong places and hence affect the user's browsing experience as well as the website's functionality. Finally, not all attributes which are used for fingerprinting can be changed (see Section 5.1) and unchangeable attributes may still be sufficient for device recognition.

Prior work has shown that data measured by hardware sensors like accelerometers can be used to fingerprint mobile devices [8, 11]. Although this is only a reasonable technique to recognize devices having such sensors, and it needs extensive training measurements. In practice, ad-trackers and anti-fraud systems gather data for fingerprints in less than a second. For our test, we also used accelerometer data, but this was measured in less time and combined with other features instead of being used as single feature.

Besides accelerometers, the internal clock of mobile devices has been investigated for fingerprinting [17, 19]. This elaborated technique is used to measure hardware imperfections of a device's internal clock by comparing to time synchronization services. Although this is a reliable method to fingerprint mobile devices, it requires measurements over a long period of time, which is not applicable for modern web-based fingerprinting. Additionally, there is no way to access a device's hardware clock via JavaScript, so calculating clock skews is not realizable using web techniques only.

Targeting mobile devices, Azizyan et al. proposed to identify the user's ambience with the help of environmental features like sound and light [5]. Although this approach may provide information about a user's location, it may not be applicable for low-noise fingerprinting through the web. A mobile browser will ask for permission to use hardware such as the microphone, which arouses the user's awareness. In contrast, we restricted our work to inconspicuous web techniques that are realizable in practical scenarios.

7 Conclusion

Client fingerprinting is used in practice for several use cases like fraud detection or user tracking. In this paper, we studied tracking libraries that are established in the field and compared their methods for fingerprinting client systems. We reviewed and investigated these common methods with regard to mobile devices and discovered that attributes that are characteristic for full-weight browsers of desktop computers lose their descriptive power when applied to mobile devices.

We then introduced a feature set containing common features from the examined libraries as well as features especially targeted at mobile devices and applied weights based on the information gain of each feature. The evaluation of this feature set is based on real-world data from an online test survey and shows that mobile devices can be fingerprinted well.

Finally, we investigated how users can evade mobile device fingerprinting, to allow a users to decide whether or not she wants to be tracked. We discussed different possibilities for web users to protect their privacy and studied four different evasion scenarios. The results indicate that it is possible, although not easy in practice, to escape fingerprinting mechanisms that aim at tracking mobile devices using our advanced techniques. An attacker may be able to circumvent fingerprinting set up for fraud prevention by taking high efforts while a mobile Web user may not be able to evade fingerprinting set up for user tracking and personalized advertisement.

References

[1] Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., and Diaz, C. The web never forgets: Persistent tracking mechanisms in the wild. SIGSAC 2014.

[2] Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., and Preneel, B. FPDetective: Dusting the web for fingerprinters. CCS 2013.

[3] Alexa Internet, Inc. Top 1M Websites. `http://www.alexa.com/topsites/`, 2014.

[4] Applications, N. Mobile/Tablet Browser Market Share. `http://www.netmarketshare.com/browser-market-share.aspx`, 2014.

[5] Azizyan, M., Constandache, I., and Roy Choudhury, R. Surroundsense: Mobile phone localization via ambience fingerprinting. MobiCom '09.

[6] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Srndic, N., Laskov, P., Giacinto, G., and Roli, F. Evasion attacks against machine learning at test time. ECML PKDD 2013.

[7] Boda, K. Firegloves. `http://fingerprint.pet-portal.eu/?menu=6`.

[8] Bojinov, H., Michalevsky, Y., Nakibly, G., and Boneh, D. Mobile device identification via sensor fingerprinting. *CoRR abs/1408.1416* (2014).

[9] Brade, K. The tor browser. https://gitweb.torproject.org/tor-browser.git.

[10] Browserleaks.com. Chrome Web Store Detector, 2014. http://www.browserleaks.com/chrome.

[11] Dey, S., Roy, N., Xu, W., Choudhury, R. R., and Nelakuditi, S. AccelPrint: Imperfections of Accel- erometers Make Smartphones Trackable. NDSS 2014.

[12] Eckersley, P. How Unique is Your Web Browser? PETS 2010.

[13] Eubank, C., Melara, M., Perez-botero, D., and Narayanan, A. Shining the floodlights on mobile web tracking – A privacy survey. W2SP 2013.

[14] Gartner Inc. Gartner says emerging markets drove worldwide smartphone sales to 19 percent growth in first quarter of 2015. http://www.gartner.com/newsroom/id/3061917, May 2015.

[15] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The weka data mining software: An update. *SIGKDD Explor. Newsl. 11*, 1 (Nov. 2009), 10–18.

[16] Kamkar, S. Evercookie – never forget. Retrieved at April 29th, 2014 from http://samy.pl/evercookie/.

[17] Kohno, T., Broido, A., and Claffy, K. Remote physical device fingerprinting. *Dependable and Secure Computing, IEEE Transactions on 2*, 2 (April 2005).

[18] MaxMind, Inc. MaxMind GeoIP2. https://www.maxmind.com/en/geoip2-services-and-databases.

[19] Moon, S., Skelly, P., and Towsley, D. Estimation and removal of clock skew from network delay measurements. INFOCOM 1999.

[20] Mowery, K., and Shacham, H. Pixel Perfect: Fingerprinting Canvas in HTML5. W2SP 2012.

[21] MSDN / Microsoft. userData Behavior, 2014. http://msdn.microsoft.com/en-us/library/ms531424%28VS.85%29.aspx.

[22] Nikiforakis, N., Joosen, W., and Livshits, B. Privaricator: Deceiving fingerprinters with little white lies. Tech. Rep. MSR-TR-2014-26, February 2014.

[23] Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., and Vigna, G. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. IEEE Symposium on Security and Privacy 2013.

[24] Stone, P. Pixel perfect timing attacks with HTML5. *Context Information Security (White Paper)* (2013).