User Tracking on the Web via Cross-Browser Fingerprinting[1]

Károly Boda[1], Ádám Máté Földes[1], Gábor György Gulyás[1], Sándor Imre[1]

[1] Department of Telecommunications, Budapest University of Technology and Economics, Magyar tudósok krt. 2., H-1117 Budapest, Hungary
bodakaroly88@gmail.com, {foldesa, gulyasg, imre}@hit.bme.hu

Abstract. The techniques of tracking users through their web browsers have greatly evolved since the birth of the World Wide Web, posing an increasingly significant privacy risk. An important branch of these methods, called fingerprinting, is getting more and more attention, because it does not rely on client-side information storage, in contrast to cookie-like techniques. In this paper, we propose a new, browser-independent fingerprinting method. We have tested it on a data set of almost a thousand records, collected through a publicly accessible test website. We have shown that a part of the IP address, the availability of a specific font set, the time zone, and the screen resolution are enough to uniquely identify most users of the five most popular web browsers, and that user agent strings are fairly effective but fragile identifiers of a browser instance.

Keywords: web privacy, user tracking, user identification, profiling.

1 Introduction

In the very beginning of the creation of the Web, users could be effectively identified by the IP addresses of their computers [2]. Later on, as the use of dynamic IP addresses and Network Address Translation became widespread, this piece of information alone was no longer enough; instead, tracking the browsing habits of a user could be performed by storing an identifier in a cookie in the web browser, so that it would supposedly identify the user for every HTTP response containing the cookie. This technique has two significant disadvantages: the cookie can only identify a single browser application, and the cookie database can be wiped, destroying the identifier. However, these techniques still seem to be widely used, albethey somewhat aged [9], [10].

Although there are cross-browser storage techniques that avoid the aforementioned problem (e.g. Local Shared Objects or LSOs, also known as Flash cookies [6]), active tracking methods (i.e. those that rely on client-side storage) all share the shortcoming of the possibility of destroying the identifier, which fueled the research of passive methods. These techniques do not store anything on the user's computer; instead, they

[1] This is a draft version. The original publication is available at www.springerlink.com.

query certain parameters that are accessible through the web browser, e.g. time zone and screen resolution.

Passive techniques include history stealing attacks [3], [4], and browser fingerprinting algorithms [1]. With history stealing, the attacker website tries to extract unique history entries from the browser – usually by exploiting unpatched vulnerabilities or misusing API functions. Browser fingerprinting checks certain properties of the browser and the computer it is run on, and tries to calculate a unique identifier from the gathered information. The first major fingerprinting experiment was Panopticlick [1], which has amassed more than 1.5 million records; it identifies users based on the so-called User Agent String (UAS, i.e. a line of text that includes the most important information about the system and the browser), parameters from the HTTP request, the list of plugins, the time zone, the screen resolution, the set of installed fonts, and the availability of some cookie-like storage techniques.

Switching browsers might provide some protection against fingerprinting, but it is unlikely that somebody would install several versions of multiple browsers to avoid being tracked. However, one version each of multiple types of browsers installed on a single computer is not uncommon. (Although defeating tracking techniques is presumably not the main motive; reasons could range from platform-exclusive extensions to selectively optimised webpages.). Furthermore, browser extensions that allow spoofing certain settings (e.g. the UAS) can also be effective means of defence. It must be noted, however, that these measures are completely ineffective against cross-browser fingerprinting techniques, which rely on other parameters, for instance, on the detection of installed font types or plugins. In order to cross-browser fingerprint a user, a website operator has to choose some browser-independent features as a basis of identification. These are likely to include a set of, but are not limited to, the following browser- and system-dependent properties:

- Networking information. Since HTTP requests are sent via TCP/IP, the server always sees the IP address (and hostname), and the TCP port number. The location of the client can also be inferred from the IP address in most cases.
- Application layer information. The user agent string is a standard HTTP header, and is sent with every request. It contains the type and version of the browser; the name and version of the operating system; the type and version of the layout engine (e.g. Gecko for Firefox); and the names and versions of certain extensions. It must be noted that some browsers (e.g. Opera) are extremely verbose about their version, so even minute patches change their UASes. Finally, the HTTP request usually contains a language preference code (e.g. ‘en-us’), too.
- Information gained by querying the browser. JavaScript programs have access to the list of fonts, plugins (along with their version numbers), screen resolution, and the time zone. Additionally, some vulnerabilities may allow access to browser history [3] or to other client-side databases that are otherwise inaccessible for the visited website.

In this paper, we discuss a new browser-independent fingerprinting technique as our main contribution, and provide the analysis of the collected data in regard to a related experiment. Our most important contribution is the analysis of font detection

via JavaScript from the viewpoint of using the detected fonts as input for fingerprinting. Furthermore, we analyse other browser-related information sets, such as the UASes, which were eventually not incorporated into the aforementioned fingerprinting algorithm, but were collected during the same experiment; we have shown that these may also be of interest for a tracker with different goals than ours.

The paper is structured as follows. In Section 2, we briefly discuss the evolution of techniques that aim to track the browsing habits of users. Then, in Section 3, we describe our own browser fingerprinting experiment, and compare the gathered results to the Panopticlick dataset. Subsequently, in Section 4, we analyse the results which we collected by it. Finally, we discuss improvements to the algorithm in Section 5, and then conclude our work in Section 6.

2 Brief History of Web Privacy

The goal of tracking one's browsing habits is usually to amass as much information about her from as many sources as possible. Such an extensive collection of data about somebody is called profiling.

Profiling has several approaches [11]. First, 'information superpowers' provide an extensive set of services (e.g. mail, calendar, social network) that cover most needs of average users, effectively making them profile themselves; Google can be taken as an example. Second, publicly accessible sources of personal data (e.g. social networks, microblogging services) can be used to get information about somebody with minimal effort. Finally, one can use tracking techniques, which comprise cookie-based methods and fingerprinting alike.

The technique of evercookies [5] combines multiple persistent storage spaces of the browser – including traditional HTTP cookies, LSOs, and HTML5-based databases – to mark the browser with a virtually indestructible identifier. If the evercookie script detects the absence of the identifier from any of the exploited storage spaces, it can recreate any of them from the remaining ones. The method can track the user across browsers hosted on the same computer if plugins that are shared by multiple browsers are installed (e.g. Adobe Flash and Microsoft Silverlight). However, evercookies are still vulnerable to deleting local storages, and furthermore, cross-browser storages may not be supported forever.

Another, local storage-independent, but more or less defunct technique of identification is history stealing [8]. It checks for the presence of popular websites in the browsing history, and identifies the user with the set of hits. In order to do this, the tracker embeds a set of invisible hyperlinks into a website, and queries their colours from a script. By default, all popular browsers mark already visited links with a different colour from that of uncharted ones. However, current browsers (e.g. Firefox 5.0) no longer allow checking such properties of hyperlinks from JavaScript code; therefore, CSS-based history stealing does not seem to be an effective means of tracking anymore [7]. But stating that the history stealing-like attacks are completely finished is untimely: besides other methods [12], history stealing inspired the algorithm of 'whitelist stealing', which works with the combination of Firefox and the

NoScript plugin, and guesses whether well-known sites have been whitelisted by the user or not [4].

To the best of our knowledge, the Panopticlick fingerprinting experiment is the first major attack in the literature [1]. This project has a publicly accessible webpage[2] where users can generate their fingerprints and get an understanding about how unique their software configuration is. More than 1.5 million users have submitted their browser fingerprints to the database so far. The identification is based on the UAS, the ACCEPT header of the HTTP request, the screen resolution and colour depth, the time zone, the presence and, if applicable, the version number of plugins and extensions; the list of fonts available on the computer; information whether JavaScript is enabled or disabled; and information whether certain persistent storage requests ('supercookies') are accepted by the browser.

Panopticlick has several merits. First, its runtime – which manifests itself only on the client side – is minimal. Secondly, it does not rely on persistent storage techniques, which means that even disabling all cookies is an ineffective defence measure. And finally, it is extremely precise, according to their results – browsers enabling Flash or Java can be identified at an accuracy rate of 94.2% [1].

The disadvantages of Panopticlick include being dependent of the browser instance. The reason for this is that the UAS and the complete font and plugin lists are taken into consideration. Moreover, either Adobe Flash or Java must be enabled to query the font list. To avoid this shortcoming, we propose a novel method, a cross-browser fingerprinting algorithm that is based solely on JavaScript, and incorporates font detection as its core technique. Our preliminary experiments showed promising results, and to prove the viability of such a technique, we started to build a fingerprint database for further analysis.

It must be mentioned that Panopticlick and our method share a common drawback, namely the inability to distinguish instances in a set of identically configured computers, as is the case when multiple users browse the web in a computer room at the same time. We argue, however, that this obstacle cannot reasonably be overcome by passive techniques; persistence is a must for identifying almost completely identical computers.

3 Harvesting Method and the Fingerprint Dataset

3.1 Collecting Fingerprints on the International PET Portal and Blog[3]

The goal of our fingerprinting technique is to create a unique identifier from specific browser data using JavaScript and server-side algorithms. Therefore, the following factors are taken into consideration: *system features* (e.g. operating system, screen resolution), *list of installed font types*, and the *first two octets of the IP address*. The chosen system features can be easily queried from the well-known global variables,

[2] http://panopticlick.eff.org/
[3] http://pet-portal.eu/fingerprint/

but to acquire the list of installed fonts, we use custom JavaScript to test each font in a given database if it is installed on the client machine.

We wanted to achieve the following goals: browser independence, plugin independence, and cross-domain tracking. Browser independence is granted by using general attributes, and we found a set of fonts that makes the font detection algorithm work equally well in all major browsers. Since we do not check installed plugins, our technique is plugin independent (for plugin detection see Section 4.6). Finally, cross-domain tracking is achieved by using only the first two octets of the IP address, which remains constant in many cases even if the IP address of the client changes dynamically. It must be noted that this concept may not work in general, as the first two octets may change after switching to a different ISP or another service (e.g. from wired to 3G data) of the same ISP; the success depends on many factors (e.g. the size of the country where the user resides or the assignment scheme of IP addresses for that country). However, this is a relatively small source of entropy for our generated user ID, and may be left out in future experiments. We do not further discuss this possibility in this paper; we leave it as future work.

Table 1 shows a list of attributes in the database structure. The meaning of most variables is clear, but there are a few to explain. The first one is 'basic fonts'; it is a specialized font list which contains thirty selected fonts that can be recognized in all major browsers (this is included in the fingerprint). The user ID is the script-generated identifier, derived from the first two octets of the IP address, the screen resolution, the time zone, and the 'basic fonts' variables; we stored its hash under the name 'short user ID'. We also store the full font list for researching a better feature set.

Table 1. Database structure: field names and their content.

locality	Hungarian or international
short user ID	user ID in a shorter, hashed format
created	time of fingerprint creation
ip	visitor IP address in a hashed format
UAS	the user agent string of the browser
os	operating system
screen	screen resolution
timezone	time zone
basic fonts	standard font list for user ID generation
all fonts	all detected installed font list stored for analysis

Later during the research, we extended this structure with some extra variables in order to be able to make more complex queries, and get more detailed results. We added 'universal font list', which is derived from 'all fonts', but it contains only those fonts that are recognised equally well by all browser versions and types (and therefore could be included in a future feature set). Finally, we also added an extra browser version field called 'browser version', derived from the 'user agent string'. It has two parts; one is the browser name (Firefox, IE, etc.), the other is the major version

number (3.6, 3.9, etc.). This field helps to make simpler and more precise browser-specific queries.

3.2 Database Statistics

The dataset under examination was collected for six months from September, 2010. For each user who clicked the 'Start the test: I want my fingerprint!' button, we recorded all data above, and displayed the fingerprint generated by the server-side script.

During the six months' data collecting process we had 989 test runs from 615 different IP addresses, which generated 662 different user IDs. These numbers suggest that most of the identifiers have been created by different computers. However, there are some IP addresses which belong to more than one ID, which can occur in two situations: either the same tester tried various settings or browsers, or there were multiple testers who connected with the same IP address (we conjecture that the former is a more likely occurrence).

The database allowed us to make some interesting observations. There are 450 different UASes, which means that nearly half of the browsers in our database are unique just by the UAS. The number of different font lists (i.e. all fonts) is 588, which suggests that it is a quite unique identifier for browser instances. There were 649 visitors with Hungarian national, and 340 visitors with international language settings. This information is not enough to infer anything significant, but it should reflect the proportions of the nationalities of our testers.

Before the analysis, we assumed that the font lists are in conjunction with the operating system; to prove this, we ran a specialized query. The result is that for Windows and Mac, the number of different font lists are close to the number of different user identifiers (slightly in excess of it in the case of Windows), and for Unix-like systems, the number of different font lists is much smaller than the number of different user identifiers. This suggests that, in the case of Unix-like systems, the font list is not a relevant factor, in contrast to other OSes.

The testers used 68 different screen resolutions; the most popular was 1280x800 with 18%, followed by 1280x1024 with 16% of the testers. The majority of the testers used a widescreen resolution, especially specific to notebook screens. The count of the most popular combination of (basic fonts, screen resolution, time zone) is 26 from 9 different IP addresses, which is a quite small set to maintain anonymity.

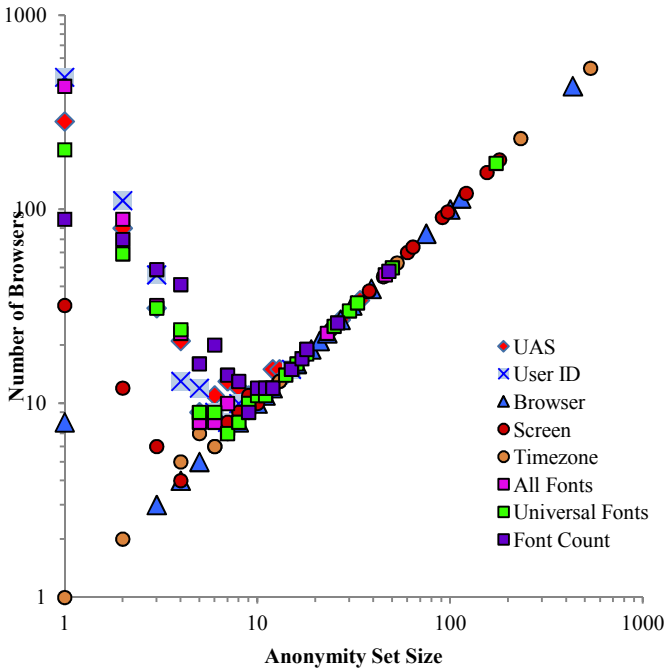

Fig. 1. The number of identical values for different attributes.

The script ran in 26 different major browser versions the preferred one being Mozilla Firefox 3.6 with 44% of the tests. The tests showed that the system- and browser-related features can greatly vary; we depicted these on Fig. 1. (We would like to mention that the graph is very similar in shape to that of the Panopticlick project [1].) After analysing the dataset, we can state that 72% of the users tested the fingerprinting algorithm only once, and, within the group that tried multiple times, 28% certainly tried multiple types or versions of browsers.

3.3 Comparison with the Panopticlick Dataset

Since the number of entries in our fingerprint database is relatively limited, we verified the quality of the data by comparing the entropy of the attributes of the entries to their counterparts in the Panopticlick dataset [1]; the current values of the entropy of the latter are publicly available on their website. The comparison is summarised in Table 2. The fields marked with an asterisk were added during the analysis of our dataset (when comparing the results, it must be taken into consideration that the Panopticlick database had more records than ours[4]).

[4] We compared the datasets in April-May, 2011.

Table 2. Comparison of the entropy of values in our database and that of Panopticlick.

	Our	Panopticlick
User agent string	8.095	10.0
Timezone	2.22	3.04
User ID	9.03	-
All fonts	8.57	13.9
Universal fonts*	6.83	-
Detected fonts*	7.63	-
Plugins	-	15.4

We would like to highlight two variables having high entropy: these are the user IDs in our experiment, and plugins in that of Panopticlick. Based on the comparison of the two values, we conjecture that the entropy of the user ID could be even higher (around 12-14 bits) for a bigger data set. The current fingerprint in our experiment does not incorporate the 'plugins' variable due to the fact that we did not attempt plugin detection; in the future, it might prove to be advantageous to include it, too. Furthermore, according to the entropy of the fonts in the Panopticlick database, a more precise user ID generation could be done with a greater dataset (and with more font types included).

4 Lessons Learned: Analysis of the Dataset

4.1 Capabilities of the New Fingerprinting Method

An interesting finding in our research was the correlation of font lists and UASes: the results show that the font lists provide a solid base for unique identification under the Windows and Mac OSes (see the graph on the left-hand side of Fig. 2). We have made another important discovery on the correlation of UASes and the generated user IDs, even though the latter did not incorporate the UAS (only the OS type). This means – if we assume that UASes are more or less unique per user – that the generated user IDs can also be assumed to be a precise identification method for all OSes (see graph on the right-hand side of Fig. 2 and Section 3.2).

Therefore, similarly to the algorithm of Panopticlick, it seems that our method can efficiently track changes in the dynamic IP address (e.g. when a user reconnects or roams between networks), and distinguish between different PCs behind a NAT (e.g. computers in an office). However, in contrast to the method of Panopticlick, our fingerprint is resistant to updates to the computer and the browser; switching browsers, (un)installing plugins, and, due to the exclusion of the UASes, emptying local storage spaces are ineffective against an attacker using it, as is explained in Section 3.2.

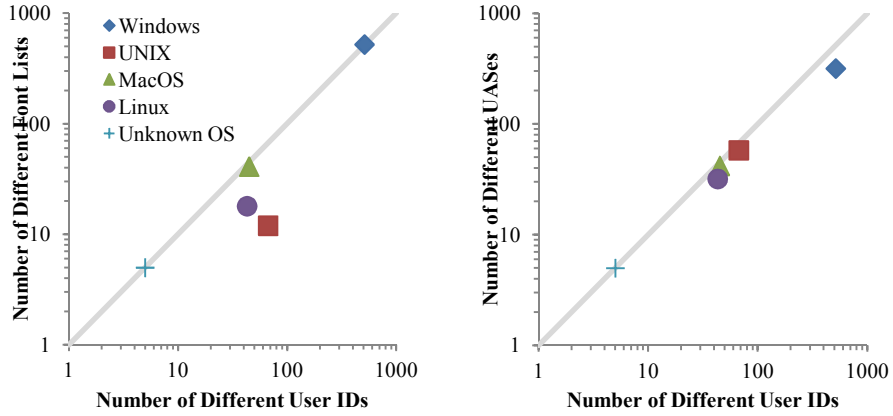

Fig. 2. Correlation between the number of font lists, UASes, and generated user IDs for different OSes.

However, there are several important lessons learned from the analysis. We have discovered system- and browser-specific weaknesses: we have identified a number of operating system-specific weak spots, e.g. the verbosity of the .NET framework version number and the massive quantity of different fonts under Windows, and the detailed version number of the OS under Linux (see Table 3 for examples). Some browser-specific weak spots also facilitate fingerprinting, the verbose browser build codes and toolbar version numbers being the most characteristic examples.

Through the analysis of the UAS, we have got to know the properties of its components, and concluded that it was a unique identifier for many users (see Section 4.3). We have determined, for each browser version, the degree of uniqueness of the UAS, i.e. the percentage of unique UASes for further analysis. Besides, the font list is also quite unique; the introduction of a universal font list (i.e. a list of fonts that a browser is limited to choose from for rendering) could make fingerprinting harder (see Section 4.4).

Our results have allowed us to define a new combination for better identification. We have got to know that the various identifiers, e.g. user ID, UAS, IP address, and font list, can be used to track changes in the fingerprint, largely due to the correlation between them being almost zero for the most popular OSes. For instance, if one of these changes while the other three remain constant, then the signatures pertaining to a user can be grouped together even if they are in an office behind a NAT router (see Section 4.6 for some proposals). However, we have not yet written a proof-of-concept program for this idea, and leave it as future work.

4.2 Special Cases

Fingerprinting allows the correction of failures in identification in some cases. By visual inspection, we have successfully identified some of the returning visitors, who:

- used more than one browser,
- tried multiple combinations of screen resolution and time zone,
- or changed their IP addresses dynamically.

Besides comparing different attributes in the records, the stored UASes helped also in matching these records, being sources of additional entropy. Based on these observations, we can track the changes in attributes, naturally within certain limits; these correction techniques can be automatised, but this is out of the scope of this paper. (We would like to mention that the Panopticlick project defined a very simple correction technique that would allow precise monitoring of changes in fingerprints [1].)

However, there were certain errors, i.e. where the algorithm produced failures, indicating weaknesses to be improved in future research. Here we give three examples of these. The first one was a user getting multiple IDs after trying multiple browsers. There were 28 such cases, all of which can be accounted to erroneous font detection, i.e. not all fonts being browser independent. The use of the universal font list decreased this number to 6, which is the same result as in the second type of error; therefore, this can be corrected by using fonts for fingerprinting from the universal font set.

In the case of the second type of error, despite most factors being identical in the analyzed records, different fonts (and OSes) were detected, resulting in different IDs. This was the issue for 6 records, from which 4 had identical UASes, 5 of them had Linux, and 1 had Windows 98 (allegedly). However, this does not necessarily mean an error, e.g. the user might have installed some fonts between the two fingerprinting attempts, and/or used a tool for spoofing the UAS.

For the third type of error, everything was identical within the records (including the UAS and the user ID), but the algorithm detected different basic fonts (used in the fingerprinting feature set). Despite our efforts, we could not reproduce this phenomenon.

4.3 Anonymity Sets

The sets of different identifiers can be distributed into many small anonymity subsets. In the following, we examine the anonymity sets of the current font lists and UASes, and the means of influencing them under the current circumstances. Unfortunately, we have a large number of small subsets in each identifier. On the graphs, we denoted with AF (Arbitrary Fonts) if clients were allowed to use arbitrary fonts, and with UF (Unified Fonts) if they were allowed to only use a limited set of fonts (however, the viability of using a unified font set on the web may be questionable).

Figure 3 demonstrates how much the anonymity set of the ideal UAS alone (i.e. with JavaScript off) grows in contrast to the current UAS. It is pleasantly surprising to see that, when using the ideal UAS with JavaScript on but no font detection, the anonymity sets are bigger than with the current UAS and JavaScript off.

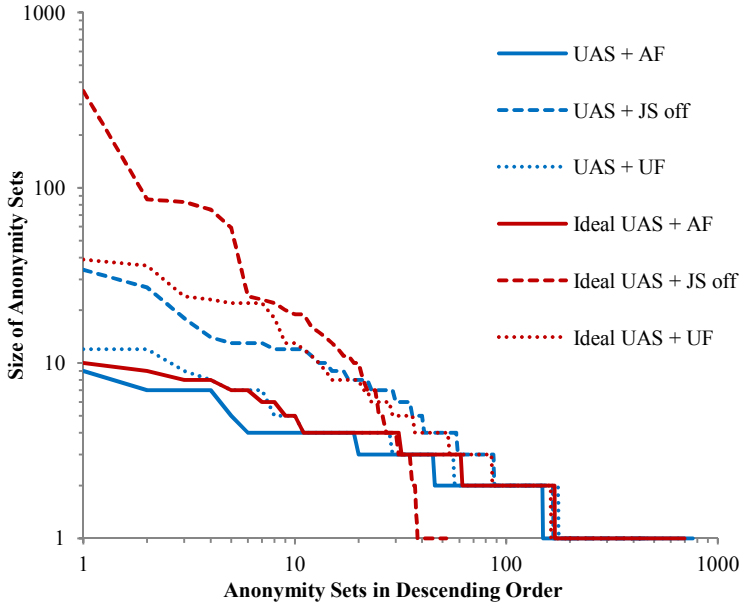

Fig. 3. Anonymity set sizes in decreasing order for different UASes with different parameters, and for different font sets (no fonts detected if JavaScript is turned off).

In the case of Linux users, the ideal UAS has a bigger impact on the anonymity sets, since the UAS does not incorporate the entire operating system version string (see the UAS analysis for more details), which results in greater anonymity sets. With JavaScript on, the situation is much worse: the curve is almost flat, i.e. all from the first 10 anonymity sets contain only a single user.

In reality, the UAS is much too detailed, and the variety of font combinations is abundant. Figure 3 also describes the anonymity sets of the current UAS with AF, UF, and JavaScript off. It can be seen that the introduction of a universal font set would improve the results, but, for now, JavaScript off results in the largest anonymity sets. Under ideal circumstances (i.e. an uncommunicative UAS and either a universal font set, or a completely disabled font detection), the results are better than with the current UAS and JavaScript off.

4.4 Font Lists

It is important to mention that in our test we also included some special font types especially used by graphic designers, not only the widespread ones. There were a few tests with these special font types, too, which may have affected the results.

We have plotted the frequency of the different font types used by the testers in descending order, and it gives a cascade curve (see Fig. 4). The source of the font types (i.e. the application whose installation package includes them) suggests that each interval between adjacent dips in the graph characterises a software package.

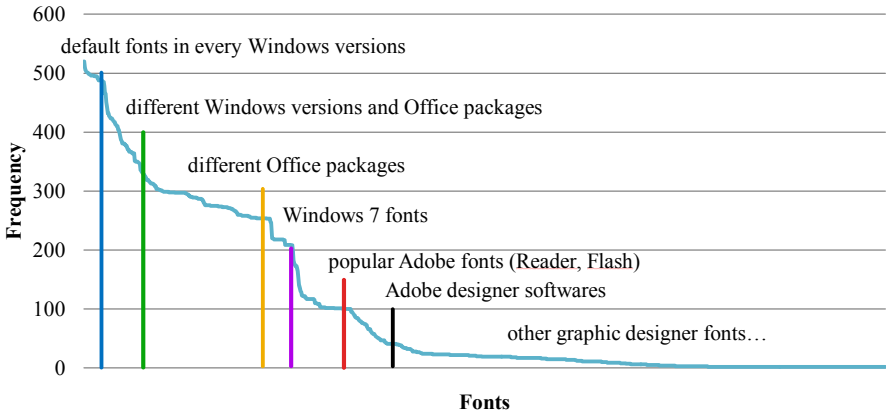

Fig. 4. Number of different font lists in descending order of the latter. (Under Windows OS)

There were multiple cases where, if the types of browsers for one tester differed, so did the font lists. This is caused by the different browser implementations. Firefox performs font replacement: if a font type is missing, it replaces it with another one from the same family (for example, if Helvetica is missing, it uses Arial as a substitute); Opera, however, does not do this, and, what is more, it does not even recognise all fonts.

This means that font type detection is not a browser-independent technique with the inclusion of all the fonts, so we had to create a font list in which all the fonts can be recognised by the five main browsers – a font list that is a partial feature set for the fingerprinting. This is what we call 'universal font list', which is created by listing the top recognized fonts by all browsers in descending order.

Eckersley suggests that browsers should only confirm the existence of fonts and plugins rather than enumerating all in a list, and queries should be answered with an exponential backoff in order to prevent exhaustive searches by malicious JavaScript code [1]. However, this would not work against the discussed font detection algorithm, since it checks the offsetWidth and offsetHeight DOM variables instead of the mere existence of a font in the system; as such, the only effective defence would be to remove these attributes from the JavaScript API.

4.5 Analysis of the User Agent String

While collecting our dataset, we recorded some browser-related values that eventually did not become parts of the cross-browser user identifier. The UAS is a highly interesting example of these, since its structure is complex (see Fig. 5 for an example), and this diversity leads to a very large number of possible combinations. It can be seen that verbosity can be increased by certain plugins, which further increases the uniqueness of the UAS.

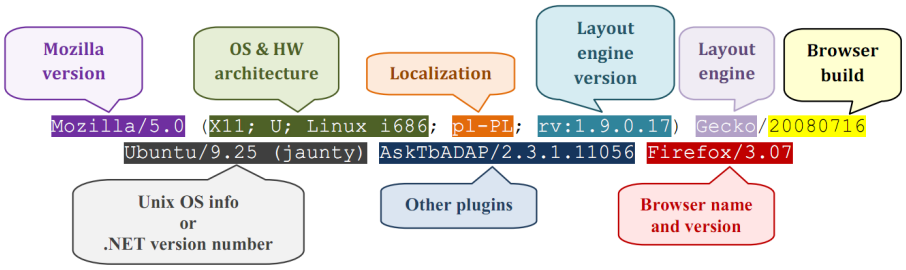

Fig. 5. Structure of the UAS.

Based on the list of recorded UASes, we can state that the most popular browser is Firefox 3.6, which sends a different UAS in 40% of its total number of occurrences in our group of testers; this is a very high percentage. For Windows systems, a major distinctive factor is the version number of the .NET framework because of its verbosity. The same versions may also differ in appearance either because of a space character before the dot, or due to multiple version numbers being shown, as you can see at Table 3. Moreover, there are some cases when a plugin or toolbar causes the difference between the same versions.

The browser build dates can also cause differences. Opera is updated so often that the version number makes a lot of browsers unique. Similarly, nearly 70% of Linux users are unique by the system's distribution name and version number, which are known to be frequently updated. Many different variations are caused by the combination of the distribution version, browser build date, layout engine version, and plugin and toolbar versions.

The UAS could be shortened easily by using less detailed version numbers, thereby decreasing the probability of its uniqueness. (It must be mentioned that the Panopticlick project had a similar proposal, too. [1]) For example, the UAS could only include the family of the operating system, and exclude the full version. Likewise, only the major browser version numbers, which can be one or two digits, should be disclosed. The parts that refer to the build date, layout engine and plugins could be entirely omitted. We argue that the ideal UAS would contain the following items: OS family, localization, browser name, and major version number, e.g. 'Opera/10 (Ubuntu; hu)'.

Most of the people use one browser at a time, and use the same browser at least for months, but most for years. This means that multiple users from one IP address (for example family members or office workers) can be differentiated just by the UAS if they do not use the same browser with the same version (and the same OS version, toolbars, and .NET version). Therefore, the UAS can be used as a local ID, but there are some cases when it seems to be globally unique (see Table 3 for examples). It must be noted, however, that current UASes are simpler, and therefore contain less information than the ones obtained during our database building period.

Table 3. Examples of unique attributes in UASes. (We have slightly mixed the actual data tuples, lest we violate our test users' privacy, but each value is taken from our database.)

.NET version	Toolbar version	Build date	Browser version	Linux version
(.NET CLR 3.0.04506.648)	GTB7.1	20100513	9.0.597.15	Ubuntu/10.10 (maverick)
(.NET CLR 3.5.30729)	GTB7.0	20100611	9.0.597.19	Ubuntu/9.10 (karmic)
(.NET CLR 3.5.30729)	AskTbPLTV5/3.8.0.12304	20100722	9.0.597.44	SUSE/3.6.12-0.7.1

4.6 Tests: Private Browsing Mode and Anonymous Browsing

Using TOR or any other proxy against the fingerprinting attempt is futile. These techniques modify the IP address only, and therefore the other parameters could still be used to track the user. We have run special queries on our database to verify this, and found 9 pairs of records where everything but the IP address was identical, and the close timestamps suggested that the user did not change her location significantly. In 2 cases, the user changed another parameter in addition to the IP address, but these records clearly belong to the same user due to the changes in the IP address. There was 1 single hit where we can reasonably assume a change in the user's location based on the big difference (i.e. more than an hour) between the timestamps; we conjecture that this was caused by the same user visiting from her laptop.

In general, disabling cookies is recommended as an extreme defence measure against tracking. The private browsing feature, which can be found under a different name in all modern browsers, aims to delete the tracks of the user's browsing activity by wiping the cache, history, cookies and other local storage spaces. However, it does not protect against passive fingerprinting, because it does not disable JavaScript, and the IP address and the UAS are still visible. Therefore, unless the browser reports a unified and uncommunicative attributes, fingerprinting will work in private browsing mode, at least to some extent.

In certain newer browsers, the user may tick a checkbox under the privacy settings configuration window to signal that she does not want to be tracked; however, this setting is not enforced, i.e. the server may choose not to honour it. This may work for honest website operators, but nonetheless, it is still up to them to decide if they stop tracking the user or ignore the request.

4.7 Increasing the Robustness of Fingerprinting

There are some values that remain constant or rarely change after the installation of the system. Because of the diversity of the values, the generated user ID has the largest entropy in our database. Dynamic IP address change does not cause any problem, because we only use its first two octets, but the modification of the supposedly constant parameters (screen resolution, time zone, and basic system fonts) makes the generated ID fail. However, the changes would be detectable using the remaining data (IP address, all fonts, and UAS).

Adding plugin detection could also increase the robustness of our fingerprinting algorithm, since there are many different plugins with different versions and a variety of combinations. The entropy of plugins was outstanding in the report of the Panopticlick project, so it is worthwhile to develop a browser-independent method for plugin detection, similarly to the font list. We argue that one should rely on those plugins that are rarely updated, and not under control of the browser; some examples of these are the Java runtime environment, Silverlight and Flash.

5 Discussion and Future Work

We are aware of certain spots to be improved in our experiment, namely that we should identify users with a cross-checking identifier (e.g. a nickname or a cookie), and we should explicitly encourage testing with multiple browsers on the same computer. For this reason, connecting related records was somewhat harder, albeit successful in most cases; however, the number of multiple-browser tests is not convincingly high. In a new experiment, we would like to avoid these issues, and focus especially on cross-browser data collection. This way, the experiment would be conducted within narrower bounds, but it would be more precise, and allow explicit identification of the client. Furthermore, plugin detection would also be included to improve the fingerprinting algorithm.

Besides the aforementioned proposal, there are several other ways of improving the algorithm. For example, a hybrid technique (i.e. one that combines fingerprinting with persistence) would likely increase the robustness of our method. One could, for instance, store an evercookie with the identifier generated by the fingerprinting algorithm, and use it to identify a user if she installs some previously absent fonts in the basic font set.

The inclusion of persistence in the method would have several other advantages. First, it would increase the accuracy of the algorithm in corporate environments, where computers are configured identically. Secondly, a persistent identifier would allow us to verify the results of the fingerprinting experiment, i.e. observe how frequently identifiers change.

In order to get a picture of the characteristics of the anonymity sets, we would need a larger data sample with several different devices and systems. It would be desirable to acquire more results with international visitors, too, to get a more nuanced data set. Moreover, as mobile devices (e.g. smartphones and tablets) spread rapidly, it would be interesting to see how our fingerprinting algorithm performs on them. We expect that such devices would have a more fixed parameter set: the screen resolution normally cannot be altered on most of them, and – supposedly – so is true for the set of supported fonts. In addition to these, there are certain other parameters that we expect not to change frequently, such as the version of the Flash plugin. Considering all these aspects, we assume that the fingerprinting of mobile devices can be carried out effectively. It must be noted, however, that one could also face some obstacles; for instance, some devices could have limited JavaScript capabilities, and certain plugins, such as Flash, might be unavailable on some models.

Finally, it would be interesting to see how well users can be identified based on the history of their IDs. Let us suppose that an attacker records the 'course' of the parameters of a certain user, e.g. by means of a persistent identifier. Then, the adversary could observe some traffic on a network, and attempt to identify the victim in the mass based on a probabilistic model. This should be defined in such a way that it tolerates some change in the fingerprint, and should be independent of the presence of the persistent identifier. Such a technique could significantly boost the power of the attacker, and constitute a serious threat to privacy.

6 Conclusion

In this paper, we have presented and analyzed a new method providing effective identification of browser instances. The latter is true largely due to the fact that the list of fonts is easily accessible through the JavaScript API. Therefore, if resistance of identification is to be secured with the web browsers of today, the only option seems to be to disable such scripts. That way, one's anonymity set can be greatly increased.

Users of computers in a set of identically configured machines are likely to have the best odds against our fingerprinting algorithm. If such computers are centrally managed, it is unlikely that any of them has a unique feature based on which it could be identified. Therefore, passive methods are insufficient in such a context.

Our research has some important implications regarding the user's behaviour. We have provided recommendations about browser types and versions that are likely indistinguishable from other such browsers. Furthermore, it can be seen that resistance to identification is easier achieved with popular system settings. It must be noted, however, that these measures are not necessarily easy to carry out without causing inconvenience to the user.

We have also given some guidelines for developers to make their browsers resist our fingerprinting algorithm. In particular, the verbosity of the UAS and the plugin versions should be decreased, and a standard 'substitute font set' – preferably supported by all browsers – should be introduced. Furthermore, users should be able to set fake system properties in the browser in order to hide their real operating system, time zone and screen resolution.

Finally, we have proposed some improvements for our method, and defined certain directions for future experiments; an explicitly cross-browser experiment would provide a high-precision dataset.

Acknowledgement

We would like to thank the Eötvös Károly Institute (http://www.ekint.org) for supporting our research. Furthermore, we would like to thank the High Speed Networks Laboratory for financially supporting our work.

References

1. P. Eckersley, "How unique is your web browser?", Proc. of the 10th international conference on Privacy enhancing technologies, Springer-Verlag Berlin, 2010, pp. 1-18, doi: 10.1007/978-3-642-14527-8_1.
2. G. Gulyás, R. Schulcz, and S. Imre, "Comprehensive analysis of web privacy and anonymous web browsers: are next generation services based on collaborative filtering?", Joint SPACE and TIME International Workshops 2008, Trondheim, Norway, June 2008.
3. G. Wondracek, T. Holz, E. Kirda, and C. Kruegel, "A Practical Attack to De-anonymize Social Network Users", Proc. of the 2010 IEEE Symposium on Security and Privacy, 2010, pp. 223-238, doi: http://doi.ieeecomputersociety.org/10.1109/SP.2010.21.
4. K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham, "Fingerprinting Information in JavaScript Implementations",W2SP 2011: Web 2.0 Security and Privacy 2011, 2011.
5. evercookie – virtually irrevocable persistent cookies, http://samy.pl/evercookie/, retrieved on August 3rd, 2011.
6. A. Soltani, S. Canty, Q. Mayo, L. Thomas, and C. J. Hoofnagle, "Flash Cookies and Privacy", 2009. Available at SSRN: http://ssrn.com/abstract=1446862.
7. Mozilla Firefox 4 Release Notes, http://www.mozilla.com/en-US/firefox/4.0/releasenotes/, retrieved on August 5th, 2011.
8. Jeremiah Grossman: I know where you've been, http://jeremiahgrossman.blogspot.com/2006/08/i-know-where-youve-been.html, retrieved on August 5th, 2011.
9. J. Gomez, T. Pinnick, and A. Soltani, "KnowPrivacy", Technical Report 2009-037, University of California at Berkeley, 2009.
10. What They Know – WSJ, http://blogs.wsj.com/wtk/, retrieved on August 5th, 2011.
11. T. Paulik, Á. M. Földes, and G. Gy. Gulyás, "Blogcrypt: Private Content Publishing on the Web", Fourth International Conference on Emerging Security Information, Systems and Technologies, SECURWARE 2010, Venice, Italy, July 2010.
12. Z. Weinberg, E.Y. Chen, P.R. Jayaraman, C. Jackson, "I still know what you visited last summer", Proc. of the 2011 IEEE Symposium on Security and Privacy, 2011, pp. 147-161, doi: http://dx.doi.org/10.1109/SP.2011.23