Mitigating browser fingerprint tracking: multi-level reconfiguration and diversification

Pierre Laperdrix
INSA-Rennes & INRIA
Rennes, France
pierre.laperdrix@insa-rennes.fr

Walter Rudametkin
University of Lille & INRIA
Lille, France
walter.rudametkin@univ-lille1.fr

Benoit Baudry
INRIA
Rennes, France
benoit.baudry@inria.fr

***Abstract*—The diversity of software components (e.g., browsers, plugins, fonts) is a wonderful opportunity for users to customize their platforms. Yet, massive customization creates a privacy issue: browsers are slightly different from one another, allowing third parties to collect unique and stable fingerprints to track users. Although software diversity appears to be the source of this privacy issue, we claim that this same diversity, combined with automatic reconfiguration, provides the essential ingredients to constantly change browsing platforms. Constant change acts as a moving target defense strategy against fingerprint tracking by breaking one essential property: stability over time.**

We leverage virtualization and modular architectures to automatically assemble and reconfigure software components at multiple levels. We operate on operating systems, browsers, fonts and plugins. This work is the first application of software reconfiguration to build a moving target defense against browser fingerprint tracking. The main objective is to automatically modify the fingerprint a platform exhibits. We have developed a prototype called *Blink* to experiment the effectiveness of our approach at randomizing fingerprints. We have assembled and reconfigured thousands of platforms, and we observe that all of them exhibit different fingerprints, and that commercial fingerprinting solutions are not able to detect that the different platforms actually correspond to a single user.

I. Introduction

Most users customize their browsers by setting their language, setting comfortable screen resolution and fonts, adding plugins to provide features, etc. Companies and open source communities have developed a massive diversity of software components to personalize browsing platforms. However, the downside to almost infinite customization is that users' platforms are unique. This, combined with easy access to the browsing platform's details, is leading to a new kind of privacy concern called "browser fingerprinting". Eckersley [1] shows that through the collection of a small set of attributes (e.g., plugins, fonts, browser type, OS), it is possible to build a browser fingerprint that is mostly unique and stable over time. These two properties make it an excellent candidate to track users. Fingerprinting, also called the "Cookieless monster" [2], is stateless and does not store anything on the user's device. This makes protecting privacy harder than in the case of cookies, while the adoption of fingerprinting is growing.

The diversity of software components to customize browsers is the source of a major privacy problem. We argue that this same diversity, combined with multi-level software reconfiguration, provides the foundations for a counter measure to browser fingerprint tracking. The idea is as follows. A browsing platform is assembled by randomly selecting a coherent set of components (an OS, a browser, plugins, etc.). Third parties collect, via the browser, enough information about the platform's components to form a fingerprint. We then regularly reconfigure the platform—thanks to a large set of diverse components—causing a different fingerprint to be exhibited. Our approach breaks one essential property for the exploitation of browser fingerprints: their stability.

We propose an original application of dynamic software reconfiguration techniques to establish a *moving target defense against browser fingerprint tracking*. We aim at regularly changing the four most distinctive elements in a fingerprint: the operating system, the browser, the list of fonts and the list of plugins. We call these the *Diversified Platform Components*, or DPC. Our approach consists in creating DPC configurations by randomly selecting components from a large pool, called the diversity reservoir. The DPC configurations are then used to assemble and reconfigure the browsing platforms, leading to diverse fingerprints being exhibited over time.

We experiment the effectiveness of our approach to mitigate browser fingerprint tracking with our prototype tool called Blink [1]. We empirically assess Blink's effectiveness in answering the following question: can we assemble and reconfigure browsing platforms that exhibit truly dissimilar fingerprints over time? We define a novel dissimilarity metric to compare the fingerprints collected from the platforms that Blink assembles. We also run commercial fingerprinting scripts to evaluate their ability to recognize the fingerprints as coming from the same user. The results show that the browsing platforms exhibit fingerprints that are very dissimilar and can deceive BlueCava, a popular commercial fingerprinting script.

The **main contributions** of this paper are:

- A moving target defense that leverages software diversity and dynamic reconfiguration to automatically assemble diverse browsing platforms.
- A dissimilarity metric that estimates the likelihood that a fingerprinter will consider two fingerprints as coming from the same platform.
- An experimental assessment of Blink's ability to automatically assemble random, dissimilar, and valid platforms.

[1] https://github.com/DIVERSIFY-project/blink

Section II presents the background and related work on browser fingerprinting. Section III describes our approach, the threat model we address, and discusses balancing privacy and comfort through users' preferences. Section IV details our prototype, Blink. Section V presents our empirical observations, while section VI concludes this paper.

II. Background

Eckersley demonstrated that browser fingerprints are often unique and unlikely to drastically change over time [1]. For example, the fingerprint in Table I "appears to be unique" among the more than four million fingerprints collected through the Panopticlick website[2]. In this section we review the state of the art and the state of practice on fingerprinting, as well as existing solutions to prevent fingerprint tracking.

A. Fingerprinting techniques

The literature reports on many ways to fingerprint browsers. User agent and Accept headers are automatically sent to websites through HTTP headers. Other information, like the list of plugins, the timezone or the screen's resolution, are easily accessible via JavaScript through the *navigator* and *screen* objects [3]. The user agent gives basic information on the operating system, while the Flash plugin gives more precise attributes. Table I displays a fingerprint including, through Flash, the Linux kernel version and the list of fonts. According to Eckersley [1], fonts are one of the most discriminating attributes. Another technique consists in testing the JavaScript engine to identify the browser and to collect information on the OS and hardware. Mulazzani et al. use tests that cover the ECMAScript standard to find functional differences between browsers [4]. Since every major browser update brings new JavaScript functionalities, it is possible to test for their presence (or absence). Mowery et al. benchmark Javascript operations and use time differences to identify the browser and the system's architecture [5]. More hardware-focused techniques also exist. In 2006, Murdoch introduced CPU skew fingerprinting [6], which uses timing information to fingerprint a device. Mowery et al. [7] and Acar et al. [8] studied canvas fingerprinting. This technique uses the HTML5 Canvas to render an image. Different hardware or software result in slight differences in the image.

B. The adoption of fingerprinting

Fingerprinting has already been adopted by many websites to complement cookies as a means of tracking. For example, the privacy policies of Google and Twitter declare they gather such data to identify users. Google states they "may collect device-specific information", such as "operating system version" or "unique device identifiers" [3], while Twitter indicates they collect "Log data", such as "browser type" or "device information" [4]. Other companies provide third party services for purchase to easily fingerprint and track users. BlueCava is a well known actor in this domain [3].

TABLE I
Example of a browser fingerprint

Attribute	Value
User agent	Mozilla/5.0 (X11; Linux i686) Gecko/20100101 Firefox/25.0
HTTP accept	text/html,application/xhtml+xml,application/xml; q=0.9,*/*;q=0.8 gzip, deflate en-US,en;q=0.5
Plugins	Plugin 0: IcedTea-Web 1.4.1; Plugin 1: Shockwave Flash 11.2 r202
Fonts	Century Schoolbook, DejaVu Sans Mono, Bitstream Vera Serif, URW Palladio L, ...
HTTP DoNotTrack	1
Cookies enabled	Yes
Platform	Linux i686
OS (via Flash)	Linux 3.14.3-200.fc20.x86 32-bit
Screen resolution	1920x1080x24
Timezone	-480
DOM session storage	Yes
DOM local storage	Yes
I.E. User data	No

C. Existing solutions to counter fingerprint-based tracking

Here we summarize solutions to limit browser fingerprint tracking and highlight some of their shortcomings.

1) Blocking extensions: Browser extensions, such as NoScript[5], Ghostery[6] and Privacy Badger[7], can block fingerprinting scripts. NoScript only runs scripts that have been whitelisted by the user. Ghostery and Privacy Badger prevent browsers from downloading scripts from *known trackers*. However, these extensions offer no guarantees against fingerprinting because it is hard to maintain up-to-date whitelists or databases. Furthermore, privacy-enhancing extensions can be counterproductive since their presence can be detected and increases the identifying information from the browser [5].

2) Spoofing extensions: User agent spoofing extensions are supposed to increase anonymity by lying about the user agent. In January 2015, we found 23 user agent spoofers for Firefox and 15 for Chrome, in their respective markets. However, as shown by Nikiforakis et al. [2], a major problem with these extensions is that they produce inconsistent headers (e.g., an iOS platform with the Flash plugin). For example, let us consider the following excerpt of a fingerprint that contains a user agent, and the platform object collected through both the JavaScript and Flash APIs:

```
UserAgent={Mozilla/5.0 (compatible; MSIE
9.0; Windows NT 6.1; Trident/5.0)} ;
JS.platform={MacIntel} ;
Flash.platform={Mac OS 10.10.2}.
```

This fingerprint, collected from a browser using a spoofing extension, exhibits an inconsistency: the user agent says the browser is MS Internet Explorer and runs on Windows NT, but the platform objects indicate Mac OS (not Windows and cannot host an IE browser). Also, these extensions act only on the user agent header and not on the entire fingerprint, allowing cross-browser fingerprinting techniques [9] to succeed.

[2] https://panopticlick.eff.org/

[3] https://www.google.com/intl/en/policies/privacy/#infocollect

[4] https://twitter.com/privacy

[5] http://noscript.net/

[6] https://www.ghostery.com/

[7] https://www.eff.org/privacybadger

3) Use of multiple browsers: Another solution is to use multiple browsers, each one exhibits a different fingerprint. However, Boda et al. [9] showed that changing browsers is insufficient because fingerprinters can rely on OS-specific data, fonts or plugins to compute a unique fingerprint since these parameters do not evolve from one browser to the next. This is known as *Cross-Browser Fingerprinting*.

4) Tor browser: The Tor browser is a modified Firefox browser, specifically designed for the Tor network. This browser attempts to create a single configuration for all Tor users so that no individual user can be identified. By design, the Tor browser's fingerprint is unique and thus known to fingerprinters. The Tor network is not immune to threats or attackers, as shown by Wang and Goldberg [10]. To remain effective, customizability and personalization are severely hampered, supplanted by the Tor browser's mono-configuration. This is also extremely brittle since a simple change (e.g., resize the browser window) stands out among the limited number of Tor users and can be immediately identified by fingerprinters.

5) PriVaricator: PriVaricator [11] is a solution designed specifically to break fingerprint stability. It uses randomization policies so the browser can decide when to lie about key fingerprintable features, like the list of plugins or font sizes. PriVaricator succeeds at fooling some well-known fingerprinters while minimizing site breakage. However, it only addresses what the authors call "explicit fingerprinting", i.e., direct attempts to collect attributes exposed by the browser. PriVaricator does not address fingerprinting performed by plugins like Flash that give access to the complete list of fonts.

III. Approach

This section establishes the threat model we target and goes into the details of our moving target approach. We explain how we leverage software diversity to change a user's browsing platform over time and we discuss the impact this has on both fingerprints and user comfort.

A. Threat model

We aim at mitigating the exploitation of browser fingerprints to track users, which is a direct threat to privacy. As noted in section II, browser fingerprint tracking relies on the following: web browsers allow remote servers to discover sufficient information about a user's platform to create a digital fingerprint that uniquely identifies the platform. We argue that fingerprint uniqueness and stability are the key threats to browser fingerprint tracking, and in this work **we aim at breaking fingerprint stability over time**.

B. A moving target defense against tracking

We propose to automatically reconfigure a user's platform to exhibit different fingerprints over time that cannot easily be linked to one another. Figure 1 shows the elements of a browsing platform that affect the fingerprint: configuration data at different levels (HW, OS, browser); software components that are assembled at different levels (e.g., `apt-get`, browser plugins, fonts); hardware components, such as the graphics card; cross-level dynamic attributes collectable only at runtime, such as through the HTML5 canvas. Once a user starts browsing the web, these data are used to create a *fingerprint*. We say that a platform *exhibits* said fingerprint.

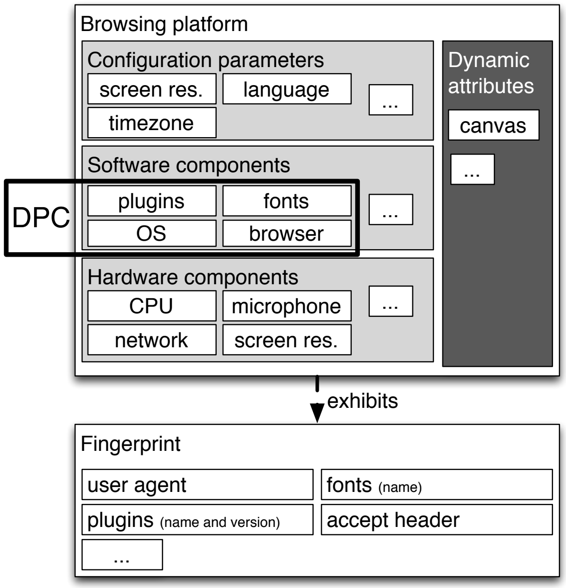

Fig. 1. User platform elements involved in web browsing and exhibited in the browser fingerprint.

Our approach reconfigures components that affect the exhibited fingerprint. Nikiforakis et al. [2] show that current commercial fingerprinters collect only a subset of all possible attributes, and Eckersley [1] found the most distinguishing attributes of a fingerprint to be fonts, plugins and user agents. Other attributes are often shared by a large pool of users, rendering them less discriminating. Based on this, we identify the following set of **Diversified Platform Components (DPC)** to be automatically reconfigured: fonts, plugins, browsers, the operating system and the CPU architecture. We call a **DPC configuration** a consistent assembly of a browser running in an operating system, with a specific set of fonts and plugins.

Definition 1: **Diversity reservoir** is the set of components used to assemble new configurations of the DPC. Given a set O of operating systems for different architectures, a set B of browsers of different types and versions, a set F of fonts and a set P of plugins, the reservoir is $DR = O \cup B \cup F \cup P$.

Our intuition is that the same set of diverse software components that cause fingerprint uniqueness can be exploited to create very large diversification reservoirs. These can be used to build trillions of distinct configurations to switch among. Figure 2 illustrates this principle: we randomly select components from the diversity reservoir to create DPC configurations used in the browsing platforms. Over time we generate new configurations that replace previous ones, changing the browsing platforms and the exhibited fingerprints. The user decides when these changes occur. This approach falls into the family of dynamic platforms, a specific form of moving target approaches, as described by Okhravi et al. in [12].

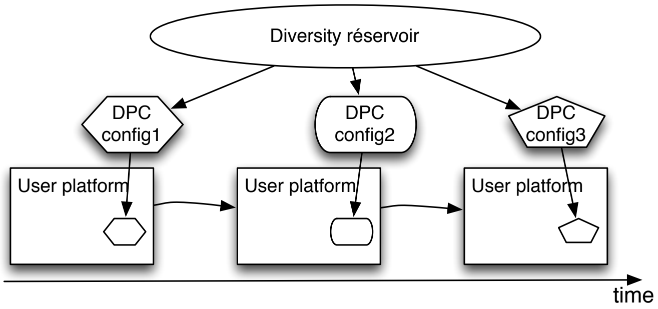

Fig. 2. Evolution of the user's platform over time.

Our moving target defense relies on an essential characteristic needed for reconfiguration: *the modular architecture of systems and browsers*. Modularity makes it possible to reconfigure the browsing platform, on demand, by automatically assembling components. This also allows us to progressively assemble configurations instead of building them beforehand.

Our approach is characterized by three essential properties: (i) the assembled platforms always exhibit *consistent fingerprints* because the platforms are genuine and we do not lie about any attributes; (ii) we assemble *correct platforms*, i.e., platforms composed of compatible components and which run correctly; and (iii) each *reconfiguration causes the exhibited fingerprints to change*. Section IV-B discusses the model that supports the assembly of correct configuration and in section V, we empirically demonstrate that variations in the exhibited fingerprints deceive two commercial fingerprinting scripts.

C. Balancing privacy and browsing comfort

There are many places where user comfort conflicts with potential gains in privacy. For example, we have included browsers in the DPC because it is an important distinguishing factor in a fingerprint. Yet, users populate their browsers with data such as bookmarks, passwords and open tabs. We believe that it is important to transfer user data between platforms we assemble in order to keep a comfortable browsing experience despite switching DPC configurations. We developed an offline cross-browser tool that is responsible for transferring the user profile between browsing platforms. We also acknowledge that not all users are ready to randomize their favorite browsers to obtain a different fingerprint, so we let them customize DPC components. Users select components from the DPC that will be included or excluded from all configurations. We call **alterable** and **essential** the components that a user decides to include, respectively discard from the set of DPC.

The choice of an essential or alterable component is subjective and can be done for functional reasons or for comfort. A set of plugins might be crucial for someone's web experience, while another user may simply wish to never change browsers. Furthermore, these choices directly impact the remaining size of the *diversity reservoir* and thus, the search space for randomization. The larger the set of *essential components* is, the smaller the total search space will be.

Deciding when to reconfigure can also impact privacy and comfort. This is a key decision because frequent changes can be effective at mitigating tracking, but are disturbing to users. However, waiting too long between changes increases the time a user's fingerprint remains static, leading to larger sessions that can be tracked. We provide two strategies for reconfiguration, a full reconfiguration strategy, called Leery, and a light-weight one, called Coffee break.

Definition 2: The **Leery strategy** reconfigures all levels of the browsing platform: the operating system, the browser, the fonts and the plugins. Essential components are kept and alterables are randomized. This strategy is used each time a user starts a new session (i.e. starts a browser for the first time). A fresh DPC configuration is generated and kept until the user ends the session. It draws its name from its cautiousness by reconfiguring as many components as possible.

Definition 3: The **"Coffee break" strategy** aims to be faster than Leery by reconfiguring only fonts and plugins. The browser and the operating system do not change. As always, essential components are kept and alterable components are randomized. This strategy is triggered in different ways: manually, by having the user lock his computer, or by waiting for a period of inactivity, hence the name.
It should be noted that reconfigurations do not occur while the user is browsing the web. To explicitly change the current fingerprint, a new session should be started (e.g., by restarting Blink) or a Coffee break reconfiguration should be triggered. Other strategies are also possible, but we feel these two cover a wide range of uses. The experimental assessment of section V-C shows that both strategies can defeat current commercial fingerprinters, but they affect fingerprints differently (more changes lead to more privacy).

D. Dissimilarity metric

To assess the effectiveness of fingerprint diversification, we need to quantify the differences between fingerprints. To our knowledge, there is no established measure to compare two fingerprints, or to determine if two different fingerprints can be related to a single platform. We define the following dissimilarity metric that aims at capturing the difference between the attribute values observed in two fingerprints.

$$D(FP_1, FP_2) = \frac{\sum_{i=1}^{8} \omega_{attr_i} \times d(attr_i(FP_1), attr_i(FP_2))}{\sum_{i=1}^{8} \omega_{attr_i}}$$

The metric is the sum of dissimilarities between 8 attributes. Its value is defined in the range [0,1]. Each attribute is weighted according to Eckersley's study [1]. Heavier weights indicate more revealing attributes. This captures, for example, that two fingerprints with the same fonts are closer than two fingerprints with the same timezone. We also defined specific dissimilarity functions d for attributes that are lists of values (e.g., fonts, user agent). The weights ω and the dissimilarity functions d are defined in Appendices A and B.

To illustrate the intuition behind our metric, let us consider two variations of the fingerprint in Figure I. If we change the browser from Firefox to Chrome and add plugins (as shown in Table II), the dissimilarity between the new and original fingerprints is 0.46. Intuitively, a third party would hardly

TABLE II
CHANGED ATTRIBUTES FOR EXAMPLE N^{O}1

Attribute	Modified value
User agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, Gecko) Chrome/34.0.1847.116 Safari/537.36
Plugins	Plugin 0: Chrome PDF Viewer; libpdf.so, [...] Plugin 12: gecko-mediaplayer 1.0.9;

TABLE III
CHANGED ATTRIBUTES FOR EXAMPLE N^{O}2

Attribute	Modified value
User agent	Mozilla/5.0 (X11; Linux i686; rv:26.0) Gecko/20100101 Firefox/26.0

consider fingerprints with different browsers and plugins to come from the same user. In the second example, only the browser version is modified (Table III). The dissimilarity is then 0.01, which indicates that the two fingerprints are almost identical. This fits our intuition: it is very likely that a browser's version will change at some point in time (e.g., it is updated). Thus, similar fingerprints according to our metric are fingerprints that are likely to be detected as originating from the same user. These two examples illustrate that some differences are more revealing than others, which is what our dissimilarity metric is capturing.

E. Positioning w.r.t. existing solutions

Our proposal is to reconfigure the browsing platform at multiple levels, causing the platform to exhibit a different fingerprint each time. It falls into the family of moving target techniques, as described by Okhravi et al. [12]. It is, by their classification, a dynamic platform, and is similar in nature to creating moving attack surfaces for web services [13]. Consequently, Blink's advantage over user-agent spoofers is to never lie. Fingerprints exhibited by the browsing platforms are based on genuine configurations, with genuine browsers, and genuine operating systems. By construction, there are no inconsistencies among the fingerprint attributes, which prevents standing out as a liar. Our approach also relates to secretless strategies because knowing the fingerprint reconfiguration strategy (random selection of DPC components) is not enough to defeat it. Cox et al. give insight into the wealth of security advantages obtained through software diversity, particularly when there are no secrets to hide [14].

Cross-browser fingerprinting shows that simply changing browsers is not enough because other attributes, such as fonts and plugins, are sufficient to create unique fingerprints [9]. We counter these attacks by randomly changing multiple levels of the browsing platform, not only the browsers.

The main drawbacks of the Tor browser are its usability and the brittleness of its fingerprint. Users should not change any of the Tor browser's configuration options, nor add plugins, because their fingerprint will diverge from the base fingerprint, disrupting the *unique fingerprint* approach and making them identifiable. Our approach is the opposite: we create unstable, always changing platforms that exhibit very different fingerprints. We welcome the addition of user plugins and fonts. And we do not force the use of any restrictive extensions (e.g., NoScript) that severely alter the browsing experience.

Finally, we do not rely on blocking fingerprinting scripts. Maintaining proper lists of scripts requires significant effort. Blink works as a moving target defense system that is oblivious to fingerprinting scripts. Blink also has the potential to resist currently unknown fingerprinting attacks thanks to the use of multi-level reconfigurations.

IV. IMPLEMENTATION

Blink reconfigures the alterable components of a DPC configuration in order to assemble unique browsing platforms. Each unique platform will exhibit a unique fingerprint because the platform components permeate the fingerprint, as seen in figure 1. Hence, by reconfiguring the platform, distinct fingerprints are exhibited. However, several issues need to be addressed for this to work in practice. Namely, the implementation must assemble and reconfigure DPC configurations that lead to platforms that function correctly. The diversity reservoir must be built up to provide a large reconfiguration space. And, finally, changing from one DPC configuration to another should be made as simple and transparent as possible.

This section describes how Blink assembles random browsing platforms, achieves a large amount of fingerprint diversity, and maintains a multi-platform user profile to improve usability. We have implemented Blink using both reconfiguration strategies described in section III-C. Those implementations are later referred to as **Leery Blink** and **"Coffee break" Blink**.

A. Multi-level reconfiguration

Blink leverages the modular architecture of operating systems and browsers to randomly assemble browsing platforms that function correctly and exhibit unique fingerprints. To maximize the reconfiguration space and the diversity of exhibited fingerprints, it is important to change as many components as possible, including the CPU architecture, the operating system, the browser, plugins and fonts, which are all statistically important for fingerprinting. Although it is possible to randomize some components directly in the user's system, changes such as adding and removing fonts can have negative side-effects on other applications. More importantly, certain components, such as the operating system or the CPU architecture, cannot be directly changed. It has been shown by Boda et al. [9] that if enough data is gathered from a system it can still be fingerprinted and tracked despite the use of multiple browsers. For these reasons, Blink uses virtual machines to maximize the diversity space that can be exploited by randomizing the operating system and the CPU architecture, all the while maintaining a high degree of isolation between the user's system, fingerprinters and the reconfiguration process.

Blink assembles components at multiple levels to form the browsing platform shown in Figure 3. We use the term *multi-level reconfiguration* because the selection of higher level components in the platform directly depends on lower level ones (e.g., a browser plugin depends on the browser, which itself depends on the operating system). Fonts are a special

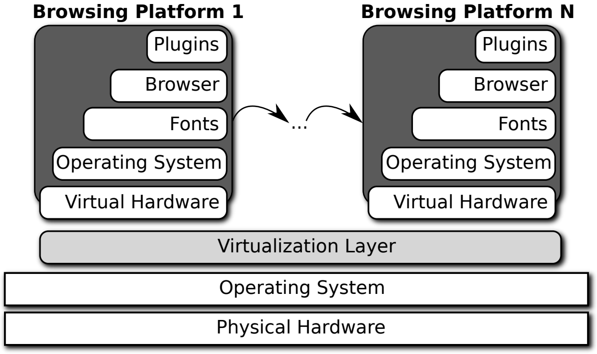

Fig. 3. A multi-level view of browsing platforms. Virtualization isolates the user's system.

case because although they do not have hard dependencies to or from other components in a DPC configuration, they do impact all user interface applications, including browsers and websites. Moreover, the modularity mechanisms are not the same for all levels: there are different dependency management tools in systems and in browsers, different plugin types and versions for different browsers, different CPU architectures, etc. Fonts may also be specific to a given system due to licensing or packaging issues.

1) Operating system reconfiguration: Blink uses VirtualBox to isolate browsing platforms because it is open source and multi-platform (supports Linux, Mac OS and Windows as both host and guest). Also, VirtualBox abstracts hardware (enabling us to randomize the CPU architecture), provides extensive functionality, has low overhead, and allows for a number of configuration parameters. Added bonuses are a user-friendly GUI and guest additions to improve integration. However, it could be substituted with other solutions.

We use VirtualBox's shared folders to transfer data between the host and the VMs. Launching a browsing platform starts the assembly of a randomized DPC configuration (i.e., OS, fonts, plugins, browser). The browser, fonts and plugins are copied to shared folders, as is the user's profile (i.e., open tabs, bookmarks, downloads, saved passwords). VMs start the `BlinkVM.py` monitoring script, whose job is to install fonts and plugins, prepare the user profile, launch and monitor the browser, and listen for commands from the host (e.g., shutdown, coffee break reconfiguration). When shutting down, shared folders are cleaned and the user-profile is recovered. VMs are additionally snapshotted when launched and rolled back to their pre-browsing states when the browsing session ends. This ensures the platform returns to a known stable state that is free of temporary files, cookies, trackers, and malware. Finally, because the OS is low-level, reconfiguring it can cause the browser and plugins to be invalidated.

2) Font reconfiguration: To install a font, the font file must be copied to the correct folder and registered. However, many applications do not detect font changes dynamically. We tested two methods to list the fonts available in browsers, namely Adobe Flash font enumeration and JavaScript font probing. Our results indicate that to ensure font changes are reflected in the exhibited fingerprint you should restart the browser.

3) Browser reconfiguration: Installing a browser requires copying the installation folder and running the main executable. Browser's run as processes so changing from one browser to another is as simple as stopping and starting them.

4) Plugin reconfiguration: Installing a plugin is as simple as copying a file to the proper plugin directory. However, unlike fonts, plugins are compiled for a specific CPU architecture and operating system, and can have runtime dependencies on configuration files or system libraries. Furthermore, not all plugins work in all browsers. We focused on randomizing NPAPI plugins (compatible with most browsers) and PPAPI plugins for Chrome (as of January 2015, NPAPI plugins are blocked by default [8]). When reconfiguring, plugins must match the CPU architecture (e.g., i686, IA-64, amd64), the operating system (e.g., Windows, Linux), and the browser (e.g., Firefox, Chrome) of the DPC to work properly. Interestingly, plugin support is often dynamic, as is the case with Chrome, Opera and Firefox. This allows Blink to reconfigure the plugin list at runtime, and is used for Coffee break reconfigurations to immediately change the exhibited fingerprint without restarting the browsing platform.

B. The diversity reservoir

At the heart of Blink is the idea that exploiting software diversity can create an ever changing browsing platform that limits attempts to track its users through browser fingerprint tracking. However, this diversity has to be built; software components must be collected and managed to ensure consistent, functional platforms are assembled and reconfigured.

1) Building the diversity reservoir: The operating system is the base of the browsing platform. We built VMs using various operating systems, although we focused on the Ubuntu and Fedora Linuxes for the i686 and x86_64 architectures. Many pre-built Linux and *BSD VMs can be downloaded freely over the internet to avoid the installation process. Other operating system's often require licenses and cannot be freely distributed. Once built, each VM must be configured to work with Blink. This includes setting up shared folders and configuring Blink's monitoring script to auto-start.

We focused on the Chrome and Firefox browsers and downloaded multiple versions of each from the vendors' websites. There's one version for each combination of release channel (stable, beta and development) and CPU architecture (i686 and x86_64). It is easy to add browsers and there are a number of forks for both Chrome (Chromium, SRWare Iron, Epic, ...) and Firefox (Iceweasel, Seamonkey, ...) that are trivial to integrate. To add a browser you must unpackage it into Blink's browser folder and follow the naming convention.

We wrote scripts to crawl the Ubuntu and Fedora repositories for fonts. We found many duplicate fonts, yet some fonts tend to be specific to a distribution. Moreover, fonts are most often installed in groups, as seen with packages that install multiple fonts. Blink allows defining font groups and system-specific fonts to avoid exhibiting uncommon fingerprints.

[8] http://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html

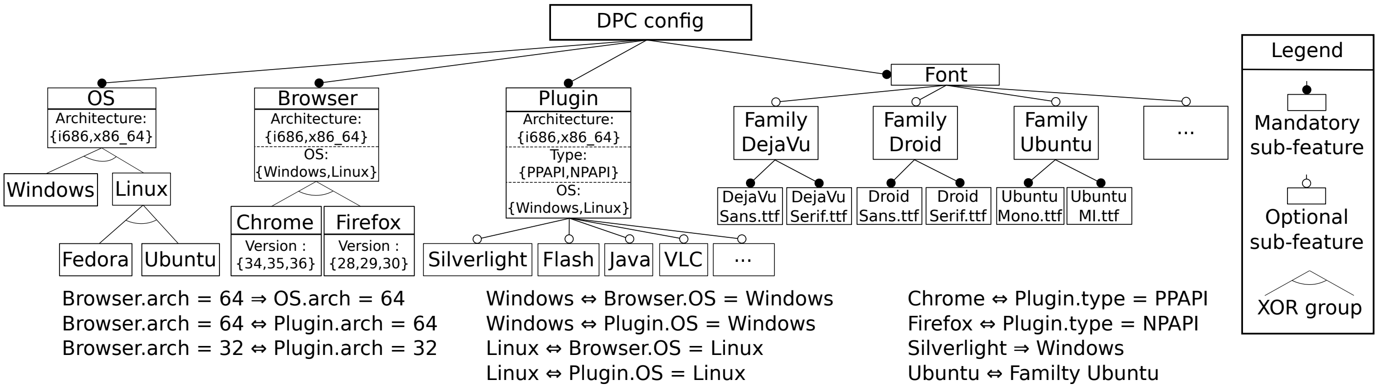

Fig. 4. An extract of the feature model used for assembling valid DPC configurations.

The biggest challenge with plugins is to account for the wealth of browser, operating system and CPU architecture combinations necessary for a plugin to function correctly in each possible DPC configuration. To obtain a large set of plugins for fingerprint diversity, we wrote scripts to crawl the Fedora and Ubuntu Linux repositories and look for plugins in all packages, for both i686 and x86_64 architectures. When a plugin is found, the package is downloaded, and the plugin is extracted. Any dependencies towards, for example, configuration files or system libraries are also extracted.

2) Modeling the well-formedness constraints in the diversity reservoir: We use feature modeling to ensure DPC configurations are valid before the browsing platforms are assembled or reconfigured. This helps establish the restrictions and conflicts among the different components in the diversity reservoir. The attributed feature model defines the valid configurations, i.e. configurations that can fundamentally run. We provide an extract of the feature model in Figure 4. A DPC configuration is composed of 4 mandatory features: an OS, a browser, plugins and fonts. It must be noted that some features in this model have attributes that specify the feature's characteristics and have domains that define possible values. For example, the attribute *Type* in the feature *Plugin* specifies that a plugin is either of type Netscape Plugin API (NPAPI) or Pepper Plugin API (PPAPI). Relations in the feature model and cross-tree constraints (see bottom of Figure 4) specify the set of rules that must be fulfilled for a DPC configuration to be valid and "runnable". For example, the first constraint states that a browser with a 64 bit architecture implies a 64 bit OS.

C. Discussion and further development

It can be argued that using virtual machines to run browsers is costly. However, current computers can easily run multiple VMs simultaneously. Many developers already use multiple VMs to target different platforms. VMs allow Blink to target multiple platforms and architectures, and to reconfigure the OS, all important discriminators in fingerprints. Moreover, the use of VMs presents beneficial side-effects: it provides very strong sandboxing for all web activity, including Firefox, which does not have a native sandbox. In our implementation, snapshots are used to return VMs to known safe-states, removing cookies and other browsing artifacts. In the end it's a tradeoff between performance, comfort and better privacy.

On the technical side, the current implementation of Blink only assembles and reconfigures Linux platforms. But, given the technical choices made, the implementation can be extended to OS X and Windows. We plan to increase the diversity reservoir by adding more browsers and OSs.

Users of our approach could potentially be spotted as having "strange" fingerprints. To mitigate this effect, we have deployed the https://amiunique.org/ web site to collect real fingerprints. So far, we have collected more than 63000 fingerprints. We observed that fingerprints rarely have more than 20 plugins and that the number of fonts range from 50 to 550. We derived two normal distributions from the data we collected and use them to set the number of fonts and plugins that we add to each DPC configuration. This way, we do not end up with configurations that would never be observed in the wild, improving the chances that Blink users do not stand out. In future work we want to exploit the insights from real fingerprints to bias the assembly of DPC configurations.

V. Empirical enquiry of multi-level reconfiguration

We present a series of experiments to **validate the effectiveness of multi-level platform reconfigurations at breaking fingerprint stability over time**.

A. Research questions

The experiments aim at answering the following questions.

RQ1. How diverse is the sequence of fingerprints exhibited by the assembled platforms? This question evaluates the ability of our approach at using the diversity reservoir to assemble series of platforms that exhibit different fingerprints. We measure the dissimilarity between the fingerprints exhibited by each consecutive pair of platforms.

RQ2. How diverse are the platforms in the eyes of actual fingerprinters? This question evaluates the ability of our approach to deceive commercial fingerprinting scripts.

B. Experiment setup

1) Our fingerprinting script: All experiments require collecting and analyzing the browser fingerprints exhibited by the assembled browsing platforms. We developed a fingerprinting script inspired by Panopticlick [1], with some improvements to gather more data via JavaScript and Flash. It works by gathering data through HTTP headers and JavaScript attributes, and it uses Flash to gain access to unique attributes in the ActionScript API. We collect the attributes listed in Table I.

2) Third-party fingerprinting scripts: BlueCava is a company that offers websites to "*recognize devices (i.e., computers, mobile phones & tablets)*" for advertising purposes[9]. One tool in their arsenal is a script that uses JavaScript and Flash to fingerprint devices. BlueCava provides a page[10] for users to opt-out of tracking that shows the identifier associated with your device. We collect these identifiers for our assembled browsing platforms. It should be noted that the identification algorithm is on the server side and is unknown to us.

Acar et al. [8] discovered a canvas fingerprinting script on the AddThis website that has since been removed[11]. The script works by sending the browser some text to render using the HTML canvas element. The text is converted into an image that is sent back to the server. Pixel variations in the image indicate differences in hardware and software used in rendering, opening the door to fingerprinting. We use the *getDataUrl()* function of the canvas element to get a URL representation of the image that allows easy comparisons.

3) Dataset: An **original platform** represents the user's browsing platform as it exists in the host operating system. The content of this platform can have a strong impact on Blink's ability at producing diverse platforms, since we import the user's fonts and plugins when assembling a DPC configuration. In essence, the plugins and fonts of the user's platform are included as essential components in the assembled platforms.

We test Blink's effectiveness using 25 original platforms. Each original platform has from 1 to 25 plugins, and from 20 to 520 fonts. The diversity reservoir for our experiments is: 4 Virtual machines (Fedora 20 32/64 bits, Ubuntu 14.04 32/64 bits); 2762 fonts; 39 browser plugins; 6 browsers, including 3 versions of Firefox: 28.0 (stable), 29.0 (beta), 30.0 (aurora), and 3 versions of Chrome: 34 (stable), 35 (beta), 36 (unstable).

4) Experimental protocol: For each of the 25 original platforms, we assemble 2 sequences of 100 platforms, providing a total of 5000. For every assembled platform, we collect the exhibited fingerprint using our script, and we collect the BlueCava and AddThis identifiers using their commercial fingerprinting scripts.

C. Results

RQ1. How diverse is the sequence of fingerprints exhibited by the assembled platforms?

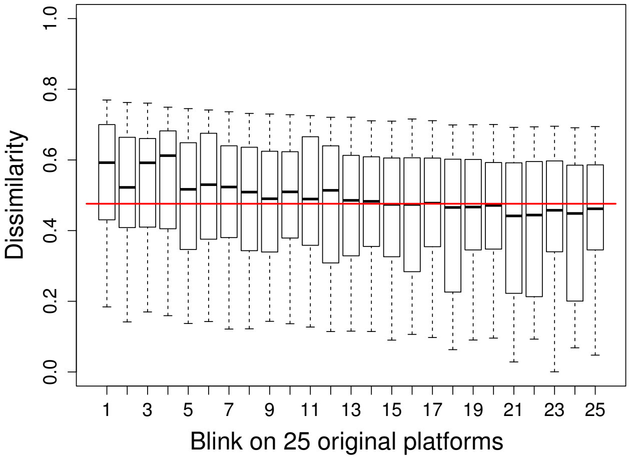

Fig. 5. Dissimilarity between consecutive platforms (Leery mode)

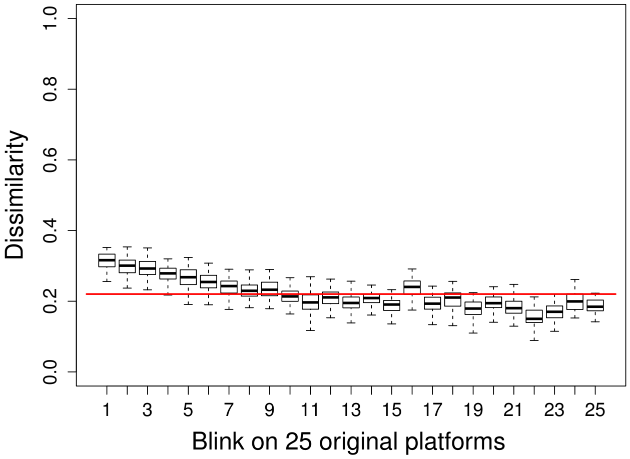

Fig. 6. Dissimilarity between consecutive platforms (Coffee break mode)

This question aims at validating that the browsing platforms we assemble exhibit different fingerprints over time. We estimate this by checking that two consecutive platforms exhibit fingerprints that are dissimilar. Thus, for each original platform, we collect the sequence of fingerprint dissimilarities by pairs of consecutive platforms:

$dissim\_seq = (D(FP_i, FP_{i+1}))_{1 \le i \le 99}$

For each of the 25 original platforms, we collect two sequences of dissimilarity values, which correspond to the 2 sequences of 100 assembled platforms. Figure 5 displays the distribution of dissimilarities in both $dissim\_seq$ for each original platform, in Leery mode. Figure 6 displays them in Coffee break mode. The X-axis in the figures correspond to the ID of the original platform, and the Y-axis represents the distribution of dissimilarities between pairs of consecutive platforms. The red line indicates the mean dissimilarity value among the 5000 collected fingerprints.

Regarding the Leery strategy, Figure 5 shows that Blink is successful at assembling successive platforms that exhibit very dissimilar fingerprints, with results up to 0.77. Blink can also assemble platforms that are very similar, with some pairs of successive platforms having dissimilarity values as low

[9] http://bluecava.com/privacy-policy/

[10] http://bluecava.com/opt-out/

[11] https://www.addthis.com/blog/2014/07/23/the-facts-about-our-use-of-a-canvas-element-in-our-recent-rd-test/

as 0.01. Since we have a relatively small pool of operating systems and browsers, it is likely that the OS or browser was the same from one configuration to the next. So, depending on randomization, consecutive fingerprints can be very similar or very dissimilar. Yet, despite frequent, large variations in dissimilarities, mean values are high thanks to our large pool of fonts and plugins.

When comparing Figures 5 and 6, the most noticeable differences are the ranges of dissimilarity. In Leery mode, dissimilarity varies between 0.01 and 0.77, while in Coffee break mode dissimilarity varies between 0.08 and 0.36. This shows the benefits of switching the operating system and browser. The mean dissimilarity values in Leery mode are more than twice as high as the ones in Coffee break mode.

Dissimilarity slowly declines as the numbers of fonts and plugins grow with each original platform. This trend occurs with both reconfiguration strategies and can be explained by the fact that the assembled platforms start to share a larger set of common plugins and fonts, lowering the overall dissimilarity score for both attributes. In the end, even if the dissimilarity is lower if we import a lot of fonts and plugins from the user's system, Blink still produces highly dissimilar configurations with an average above 0.4 in Leery mode.

We conclude from these observations that Blink generates configurations that are truly different from one another. It corresponds to our objective of breaking the stability of fingerprints over time, and shows its effectiveness in blocking the ability of fingerprinters to track users. Variations in Blink's effectiveness are due to the richness of the original platforms and on the size of the diversity reservoir.

RQ2. How diverse are the platforms in the eyes of actual fingerprinters? To answer this, we collected fingerprint identifiers from BlueCava and AddThis for each assembled platform. When these scripts assign the same identifier to two platforms, it means they consider them to be the same.

1) BlueCava browser fingerprinting script: In our tests more than 99% of the identifiers we collected were different (we had similar results for both Leery and Coffee break modes). BlueCava's script is not able to detect that the assembled platforms are from the same user, in part due to BlueCava taking into account even the slightest differences in a fingerprint. This shows that Blink is effective against unique identifiers, and confirms that Blink can deceive BlueCava, even when fingerprint diversity is low.

2) AddThis canvas fingerprinting script: From the 5000 assembled platforms, we obtained 181 different identifiers in Leery mode and 34 in Coffee break mode using the AddThis script. Blink performs relatively well despite the fact that canvas fingerprinting targets the GPU and low-level drivers. We can also see that Leery is 5 times more effective than Coffee break because the OS and browser change. Currently the DPC does not include graphical components (e.g., GPU drivers) or GPU/HTML5 specific configurations, but we plan to explore this in the future.

D. Threats to validity

To our knowledge, characterizing the impact of moving target defenses on security is an open challenge [15]. Still, this section provided empirical evidence of the effectiveness of Blink's behavior with respect to fingerprinting. We now summarize the threats to the validity of our findings.

While no established metric exists to evaluate the effectiveness of fingerprinting countermeasures, we defined our own dissimilarity metric. This is a construct threat because the metric might not properly reflect the ability of fingerprinters to decide if a fingerprint is from a distinct platform. To mitigate this threat, we based fingerprint attribute weights on the observations from Eckersley's extensive study [1]. We also collected identifiers from third-party commercial fingerprinting scripts.

The external validity lies in the ability to generalize our observations. We evaluated Blink's effect on 25 initial situations, with a pool of 6 browsers and 4 operating systems, all running Linux platforms. Starting from these situations, Blink is able to assemble platforms that exhibit very different fingerprints. Yet, we do not know how Blink behaves in other situations (e.g., users that have a very large set of essential fonts). We are exploring a new domain (mitigating fingerprint-based tracking) and further quantitative and usability studies are needed to establish a comprehensive understanding of moving target approaches in this domain.

Internal validity very much depends on the correct implementation of Blink, as well as both the fingerprinting scripts and the metrics. We mitigated this risk through thorough testing campaigns, and thousands of runs to tune Blink. Yet, there might be bugs that influence our results. We hope that they only change marginal quantitative things, and not the qualitative essence of our findings.

VI. Conclusion

This work explores the opportunity of exploiting automatic, multi-level reconfiguration and the natural diversity of software components in order to create a moving target defense against browser fingerprint tracking. We leverage virtualization and modular architectures at various levels (OS and browser) to modify, over time, the parts of a user platform that are the most identifying in a fingerprint. This completely new approach allows users to exhibit a diversity of fingerprints without lying. We also contribute a novel dissimilarity metric to evaluate the differences between two fingerprints and estimate the likelihood they belong to the same user. A tool named Blink has been developed to assess the effectiveness of this approach. Our empirical results, based on the analysis of 5000 fingerprints exhibited by randomly assembled configurations, show that Blink generates very dissimilar configurations and can deceive BlueCava, a commercial fingerprinting script.

We plan to increase the size of our diversity reservoir to generate even more diverse configurations. We have also deployed the https://amiunique.org/ website to inform users about fingerprinting and to collect a large number of real fingerprints, which will help us improve Blink.

ACKNOWLEDGMENT

We thank Nick Nikiforakis for his insightful discussions and for suggesting the Coffee break strategy, as well as Gildas Avoine and Nataliia Bielova for their expert feedback on this paper. This work is partially supported by the EU FP7-ICT-2011-9 No. 600654 DIVERSIFY project.

APPENDIX A
FINGERPRINT ATTRIBUTE WEIGHTS

Attribute	Entropy(bits)
User agent	10.0
Accept Header	6.09
Plugins	15.4
Fonts	13.9
Screen resolution	4.83
Timezone	3.04
DOM storage	2.12
Cookies	0.353

APPENDIX B
FINGERPRINT DISSIMILARITY

This annex provides the strategy to determine the dissimilarity between the eight fingerprint attributes.

[Attribute 1] User agent: We decompose the user agent attribute into two categories: data related to the browser and its version, and data related to the architecture of the device (32 or 64 bits).

$$F_{br} = \begin{cases} 0 & br.name_{FP_1} = br.name_{FP_2} \; \wedge \\ & br.ver_{FP_1} = br.ver_{FP_2} \\ 0.125 & br.name_{FP_1} = br.name_{FP_2} \; \wedge \\ & br.ver_{FP_1} \neq br.ver_{FP_2} \\ 1 & br.name_{FP_1} \neq br.name_{FP_2} \end{cases}$$

$$F_{archi} = \begin{cases} 0 & archi_{FP_1} = archi_{FP_2} \\ 1 & archi_{FP_1} \neq archi_{FP_2} \end{cases}$$

We grant equal weight to both these categories. The resulting dissimilarity between the user agents in two fingerprints is computed as follows:

$d(attr_1(FP_1, FP_2)) = 0.5 \times F_{br} + 0.5 \times F_{archi}$

[Attribute 2] Plugins: Let us consider $LP(FP_1)$, the set of plugins in fingerprint 1 and $LP(FP_2)$, the set of plugins in fingerprint 2. Plugins are stored in a list of tuples where the first element of each tuple is the name of the plugin and the second its version. We define two different sets for plugins that are common to both fingerprints: one where plugins have an identical version ($LP_{=name,=ver}$) and one where plugins have different versions ($LP_{=name,\neq ver}$).

$P \in LP_{=name,=ver}$ if $\exists P_1 \in LP(FP_1) \wedge \exists P_2 \in LP(FP_2)$, $P_1.name = P_1.name \wedge P_1.ver = P_2.ver \wedge P.name = P_1.name = P_2.name$

$P \in LP_{=name,\neq ver}$ if $\exists P_1 \in LP(FP_1) \wedge \exists P_2 \in LP(FP_2)$, $P_1.name = P_1.name \wedge P_1.ver \neq P_2.ver \wedge P.name = P_1.name = P_2.name$

The total number of plugins in both lists is computed as follows (we count the plugins that appear in both lists once):

$FU = |(LP(FP_1) \setminus (LP(FP_1) \cap LP(FP_2))) \cup LP(FP_2)|$

We calculate the proportion of plugins that are unique to FP_1 and to FP_2 as:

$$F1 = \frac{|(LP(FP_1) \setminus (LP_{=name,=ver} \cup LP_{=name,\neq ver}))|}{FU}$$

$$F2 = \frac{|(LP(FP_2) \setminus (LP_{=name,=ver} \cup LP_{=name,\neq ver}))|}{FU}$$

We get the proportion of common plugins:

$$F3 = \frac{|LP_{=name,\neq ver}|}{FU} \qquad F4 = \frac{|LP_{=name,=ver}|}{FU}$$

The dissimilarity between the plugins in two fingerprints is computed as follows:

$$d(attr_2(FP_1, FP_2)) = \frac{F1 + F2 - 0.75 \times F3 - F4 + 1}{2}$$

The dissimilarity for this attribute ranges from 0 to 1 with greater values representing an increasing dissimilarity between the two lists of plugins.

[Attribute 3] Fonts: The dissimilarity between two lists of fonts is the proportion of fonts that are only in one fingerprint, minus the proportion of fonts that are in both. This term is 0 if both lists are identical and 1 if they are completely different.

Let us consider $LF(FP_1)$, the set of fonts in fingerprint 1 and $LF(FP_2)$, the set of fonts in fingerprint 2. The total number of fonts in both lists is computed as follows (we count the fonts that appear in both lists only once):

$FU = |(LF(FP_1) \setminus (LF(FP_1) \cap LF(FP_2))) \cup LF(FP_2)|$

The proportion of fonts unique to FP_1 is:

$$F1 = \frac{|(LF(FP_1) \setminus (LF(FP_1) \cap LF(FP_2)))|}{FU}$$

The proportion of fonts unique to FP_2 is:

$$F2 = \frac{|(LF(FP_2) \setminus (LF(FP_1) \cap LF(FP_2)))|}{FU}$$

The proportion of common fonts is:

$$F3 = \frac{|(LF(FP_1) \cap LF(FP_2))|}{FU}$$

The dissimilarity between the fonts in two fingerprints is:

$$d(attr_3(FP_1, FP_2)) = \frac{F1 + F2 - F3 + 1}{2}$$

[Attributes 4 - 8] Accept Header, Screen resolution, Timezone, DOM storage, Cookies: These attributes can be compared without additional processing. The dissimilarity between these attributes is:

$$d(attr_X(FP_1), attr_X(FP_2)) = \begin{cases} 0 & attr_X(FP_1) = attr_X(FP_2) \\ 1 & attr_X(FP_1) \neq attr_X(FP_2) \end{cases}$$

References

[1] P. Eckersley, "How unique is your web browser?" in *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, ser. PETS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 1–18. [Online]. Available: http://dl.acm.org/citation.cfm?id=1881151.1881152

[2] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *Proc. of the Symp. on Security and Privacy*, 2013, pp. 541–555.

[3] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel, "Fpdetective: dusting the web for fingerprinters," in *Proc. of the Conf. on Computer & Communications Security (CCS)*. ACM, 2013, pp. 1129–1140.

[4] M. Mulazzani, P. Reschl, M. Huber, M. Leithner, S. Schrittwieser, E. Weippl, and F. C. Wien, "Fast and reliable browser identification with javascript engine fingerprinting," in *Web 2.0 Workshop on Security and Privacy (W2SP)*, vol. 5, 2013.

[5] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham, "Fingerprinting information in JavaScript implementations," in *Proceedings of W2SP 2011*, H. Wang, Ed. IEEE Computer Society, May 2011.

[6] S. J. Murdoch, "Hot or not: Revealing hidden services by their clock skew," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ser. CCS '06. New York, NY, USA: ACM, 2006, pp. 27–36. [Online]. Available: http://doi.acm.org/10.1145/1180405.1180410

[7] K. Mowery and H. Shacham, "Pixel perfect: Fingerprinting canvas in HTML5," in *Proceedings of W2SP 2012*, M. Fredrikson, Ed. IEEE Computer Society, May 2012.

[8] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The web never forgets: Persistent tracking mechanisms in the wild," in *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS 2014)*. ACM, 2014.

[9] K. Boda, A. M. Földes, G. G. Gulyás, and S. Imre, "User tracking on the web via cross-browser fingerprinting," in *Information Security Technology for Applications*, ser. Lecture Notes in Computer Science, P. Laud, Ed. Springer Berlin Heidelberg, 2012, vol. 7161, pp. 31–46. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29615-4_4

[10] T. Wang and I. Goldberg, "Improved website fingerprinting on tor," in *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, ser. WPES '13. New York, NY, USA: ACM, 2013, pp. 201–212. [Online]. Available: http://doi.acm.org/10.1145/2517840.2517851

[11] N. Nikiforakis, W. Joosen, and B. Livshits, "Privaricator: Deceiving fingerprinters with little white lies," Microsoft, Tech. Rep., 2014.

[12] H. Okhravi, T. Hobson, D. Bigelow, and W. Streilein, "Finding focus in the blur of moving-target techniques," *Security Privacy, IEEE*, vol. 12, no. 2, pp. 16–26, Mar 2014.

[13] Y. Huang and A. K. Ghosh, "Introducing diversity and uncertainty to create moving attack surfaces for web services," in *Moving Target Defense*. Springer, 2011, pp. 131–151.

[14] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser, "N-variant systems: a secretless framework for security through diversity," in *Proc. of the Conf. on USENIX Security Symposium*, ser. USENIX-SS'06, 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267336.1267344

[15] M. Christodorescu, M. Fredrikson, S. Jha, and J. Giffin, "End-to-end software diversification of internet services," in *Moving Target Defense*. Springer, 2011, pp. 117–130.