FPGuard: Detection and Prevention of Browser Fingerprinting

Amin FaizKhademi[1(✉)], Mohammad Zulkernine[1], and Komminist Weldemariam[1,2]

[1] School of Computing, Queen's University, Kingston, Canada
{khademi,mzulker,weldemar}@cs.queensu.ca
[2] IBM Research-Africa, Nairobi, Kenya
k.weldemariam@ke.ibm.com

Abstract. Fingerprinting is an identification method used by enterprises to personalize services for their end-users and detect online fraud or by adversaries to launch targeted attacks. Various tools have been proposed to protect online users from undesired identification probes to enhance the privacy and security of the users. However, we have observed that new fingerprinting methods can easily evade the existing protection mechanisms. This paper presents a runtime fingerprinting detection and prevention approach, called FPGuard. FPGuard relies on the analysis of predefined metrics to identify fingerprinting attempts. While FPGuard's detection capability is evaluated using the top 10,000 Alexa websites, its prevention mechanism is evaluated against four fingerprinting providers. Our evaluation results show that FPGuard can effectively recognize and mitigate fingerprinting-related activities and distinguish normal from abnormal webpages (or fingerprinters).

Keywords: Fingerprinting · User privacy · Detection · Prevention · FPGuard

1 Introduction

Web tracking involves monitoring and recording a user's browsing experience across different websites or multiple visits of the user to a website. Based on the recorded information, a user profile can be created for delivering personalized and targeted services (e.g., advertisements). In fact, many advertisers collect information about the users' interests, location, browsing habit, etc. without asking for their users' consent by employing either active or passive tracking methods [1]. An attentive user can prevent or permanently stop the active trackers —which store an identifier in a user's browser or computer so that they can identify the user in future visits— by just deleting the client-side identifiers or operating in the private mode of browsers. However, clearing client-side identifiers is no longer sufficient to prevent the user from being identified or tracked. The reason is that passive tracking methods such as browser fingerprinting [2] and history

P. Samarati (Ed.): DBSec 2015, LNCS 9149, pp. 293–308, 2015.
DOI: 10.1007/978-3-319-20810-7_21

sniffing [3] do not rely on client-side identifiers and hence can easily evade the existing anti-tracking tools (e.g., [4,5]).

To identify a browser, fingerprinters systematically send multiple queries to the underlying environment (through the browser) to extract various system properties such as screen resolution, list of installed plugins, system fonts, time-zone, etc. Then, they combine the collected properties to generate an identifier (or fingerprint) for the browser [2]. Fingerprinting process is invisible to users and thus they are unaware of the trackers' queries and underlying logics. In addition, unlike client-side identifiers, fingerprinting does not leave any footprint (on the browser). These features (i.e., invisibility and no footprint) make fingerprinting as a reliable source of user identification. While fingerprinting is becoming more prevalent than before [6,7], according to several studies (e.g., [8–11]), Web users prefer to have control over their privacy and thus prefer to stay anonymous to unknown or hidden tracking service providers.

To preserve user's privacy, a number of fingerprinting providers offer an opt-out solution (e.g., bluecava [12], AddThis [13]) [14]. They provide an opt-out page in which a user can choose to stay anonymous by not sharing her browser fingerprint. This solution mostly relies on server side (i.e., tracker) for collecting the browser's fingerprint, without sharing it with third-parties (e.g., advertising companies). However, the fingerprinting provider might still share the browser's fingerprint with clients who are exploiting fingerprinting for fraud detection purposes [12]. In addition, the existing anti-fingerprinting solutions [15–17] lack a detection mechanism (at runtime), therefore can disable browser functionalities for both normal and fingerprinting webpages.

In this paper, we present an approach for runtime detection and prevention of Web-based fingerprinting, named *FPGuard*. With respect to detection, FPGuard monitors running Web objects on the user's browser, collects fine-grained data related to fingerprinting activities and analyzes them to search for patterns of fingerprinting attempts. In case fingerprinting activities are detected, FPGuard notifies the user by raising an alert. It then labels the URL of the website as fingerprinter and adds it to a blacklist database as an evidence for future use. With respect to prevention, FPGuard combats fingerprinting attempts by combining randomization and filtering techniques. In particular, we employ four different randomization policies and two filtering techniques to protect users from being fingerprinted while keeping the browser's functionality for normal websites. FPGuard is implemented as a combination of a Chrome browser extension and an instrumented Chromium browser. We evaluate its detection capability using the top 10,000 Alexa websites. The prevention mechanism of FPGuard is also evaluated against four fingerprinting providers: two popular commercial fingerprinters and two proof-of-concept fingerprinters. As compared to existing approaches, our evaluation results show that FPGuard first detects and then prevents fingerprinting attempts, thereby it does not affect the user's browsing experience.

The next section discusses the related work on fingerprinting. While Sect. 3 introduces our proposed approach, Sect. 4 discusses its implementation details and our experimental analysis. In Sect. 5, we compare FPGuard with the existing work and Sect. 6 concludes the paper.

2 Related Work

There are several techniques employed for fingerprinting a browser instance with the purpose of identification. These methods mainly use especial JavaScript objects (i.e., `navigator` and `screen` objects (or *infromative* objects)) [2,14,18], Flash plugin [2], HTML5 canvas element [19], browsing history [20], performance and design difference of JavaScript engines of web browsers [19,21], IP address and HTTP accept headers [22] (accessible though network analysis) for identification. Not all of the mentioned methods exist in real-life fingerprinting products [14]. However, they can increase the identification accuracy.

With respect to detection, to the best of our knowledge, FPdetective [6] is the first large-scale study on detecting browser fingerprinting attempts by combining dynamic and manual analysis techniques. The approach focused on finding JavaScript-based font detection and Flash-based fingerprinting-related activities. The authors found that Flash-based fingerprinting (97 websites) is more prevalent than JavaScript-based font detection (51 websites) among the top 10 K websites of Alexa. They also embedded the discovered URLs of scripts and Flash objects in a browser extension, which can be used as a blacklist-based approach for finding previously recognized fingerprinting attempts. In addition to FPdetective, a large-scale study on the prevalence of canvas fingerprinting among the Alexa's top 100 K websites is presented in [7]. The authors modified the Chromium browser to log the accesses (writes and reads) to canvas elements. They combined manual and dynamic analysis to identify fingerprinting attempts and found that more than 5 % of the visited websites leveraged this type of fingerprinting.

For prevention, *Tor* [15], *Privaricator* [16], and *Firegloves* [17] are widely used anti-fingerprinting tools. Tor is a browser used to connect to the Tor network. The Tor network is an anonymous network for hiding the user's online activities and traffic by encrypting the data and passing it through a number of nodes (or relays). In addition to providing such an anonymous network, Tor is equipped with features for combating browser fingerprinting. The reason is that browser fingerprinting works well on the anonymous network and could effectively be used for identification. As explained by Perry et al. [23], Tor makes the browsers' fingerprints identical, meaning that the properties of the browser are fixed and identical for all users. In addition, it disables all browser plugins except the Flash plugin (due to high market penetration of Flash). For the Flash files, it offers a "click-to-play" option in which the user has to authorize a Flash file to be executed. Similarly, canvas elements in Tor should be authorized by users. Otherwise, the canvas elements are disabled and represent empty white images. Moreover, Tor limits the number of fonts that a website can load. In case a

document exceeds the limit, Tor does not allow the website to load more fonts. This ultimately reduces the functionality of the browser.

Privaricator enhances the private-mode of browsers [16]. It implements randomization policies on two properties of the browser by randomly changing the browser's fingerprint upon each visit to a website. Thus, every visit of a user with the same browser is considered as a new user visiting the website. However, authors only focused on the plugin list of the browser and the offset properties of HTML elements to prevent *JavaScript Objects Fingerprinting* and the *JavaScript-based Font Detection* attempts, receptively. To do so, Privaricator randomly hides a number of plugins from the browser's plugin list. Moreover, it adds noises to the offset properties of HTML elements, when the properties are looked up more than 50 times.

Firegloves [17] is a proof-of-concept Mozilla Firefox extension. It returns random values for the browser properties (e.g., screen resolution). It also disables all plugins and MIME types on the browser. To prevent JavaScript-based font detection attempts, Firegloves limits the number of available fonts on each tab and also returns random values for the `offsetWidth` and `offsetHeight` properties of the HTML elements. However, it is also possible to get the width and height of the HTML elements using the `width` and `height` properties of the `getBoundingClientRect` method. Finally, *ExtensionCanvasFingerprintBlock* [24] is a recently developed browser extension for protecting users against canvas fingerprinting by returning an empty image for canvas elements that are accessed programmatically.

Finally, as noted, the above solutions reduce the fingerprinting surface by disabling browser functionalities (e.g., diabling Flash plugin). In fact, they block both normal and fingerprinting websites as they do not have detection capability to perform prior to blocking. This ultimately will create unpleasant experience (e.g., Facebook does not load Flash resources if the Flash plugin is hidden) to users due to functionality loss of the browser.

3 Detection and Prevention Approach

In this section, we present our approach named *FPGuard* for the detection and prevention of four major fingerprinting methods discussed in previous in studies [6,7,14,25]: *JavaScript Objects Fingerprinting*, *JavaScript-based Font Detection*, *Canvas Fingerprinting*, and *Flash-based Fingerprinting*. With respect to detection, FPGuard relies on recently discovered fingerprinting metrics [25] as listed in Table 1. These metrics are indicators for fingerprinting attempts and can be used for detection purposes.

Figure 1 shows an overview of FPGuard. Given a webpage running on the user's browser, FPGuard monitors and records its activities on the user's browser from the time the webpage has started loading. Then, it extracts the metrics relative to each fingerprinting method and builds a signature for the webpage. Next, it uses various algorithms to distinguish normal webpages from fingerprinters. In case, a webpage is flagged as fingerprinter, FPGuard notifies the user with an

Table 1. Metrics that are indicators for fingerprinting attempts.

Metric	Description
Metric 1	the number of accesses to the `navigator` and `screen` objects' properties
Metric 2	the number of accesses to the properties of the `Plugin` and `MimeType` objects
Metric 3	the number of fonts loaded using JavaScript
Metric 4	the number of accesses to the offset properties of HTML elements
Metric 5	a boolean specifying whether a canvas element is programmatically accessed (writes and reads)
Metric 6	a boolean specifying the visibility status (hidden or visible) of a canvas element that is programmatically accessed
Metric 7	a boolean specifying the existence of methods for enumerating system fonts and collecting system-related information in the source code of a Flash file
Metric 8	a boolean specifying the existence of methods for transferring the collected information
Metric 9	a boolean specifying the visibility status of a Flash file (hidden, visible, or small)

alert and stores the URL of the webpage in a blacklist. The user has the option to either trust the webpage and let it run as is or prevent it from fingerprinting the user's browser using one of the designed prevention techniques. FPGuard runs in two phases: *detection* and *prevention*. In what follows, we describe these two phases in detail.

3.1 Phase I: Detection

In this phase, FPGuard identifies the fingerprinting-related activities. The core components of this phase are (see Fig. 1): *Monitor*, *Logger*, and *Analyzer*. The Monitor component is responsible for collecting browser activities. First, it injects the Logger component to the DOM tree of the webpage before the loading of other resources. The Logger component runs at the background of the browser and records all the activities of the webpage. This component then parses the logs and extracts metrics (nine metrics in total, see Table 1) for each corresponding fingerprinting method. These metrics will be used by the Analyzer component to look for fingerprinting evidences. Once the Analyzer flags the webpage as a fingerprinter, FPGuard displays an alert to the user and stores the URL of the webpage with the observed metrics into the blacklist database. Using the collected nine metrics, the Analyzer assigns a score (Score in Fig. 1) depending on the level of suspicion. Thus, we define three levels of suspiciousness for performing fingerprinting using those metrics. For example, for the *JavaScript Objects Fingerprinting* method, the Analyzer assigns three levels of suspiciousness for accesses to the informative objects (i.e., `navigator` and `screen` objects)

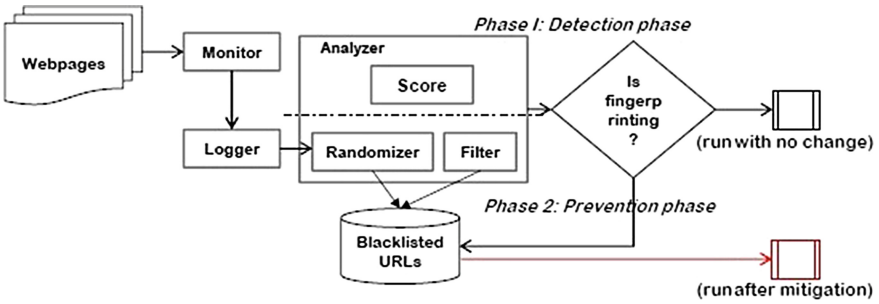

Fig. 1. An overview of FPGuard.

as well as `Plugin` and `MimeType` objects. A webpage has the first and second level suspicious access to the mentioned JavaScript objects if *Metric 1* and *Metric 2* surpass the corresponding predefined thresholds. A webpage has also third-level suspicious access when it has first-level and second-level suspicious accesses.

Similarly, for *JavaScript-based Font Detection*, using *Metrics 3 &4*, the Analyzer compares the obtained metrics against predefined thresholds and any attempt that surpasses to a given threshold value is flagged as fingerprinting. The Analyzer checks *Metrics 5 &6* and considers a canvas element as first-level suspicious, if they indicate writes and reads performed on the canvas element. The Analyzer considers a canvas element as second-level suspicious if the canvas element is first-level suspicious and is also hidden or dynamically created. Finally, the Analyzer labels each Flash file with a suspiciousness level from one to three where level three suspiciousness indicates a high probability of a fingerprinting attempt. In particular, it encodes as first-level suspicious when *Metric 7* is true. When *Metric 8* is true in the context of the first-level suspicious Flash file, then we consider it as a second-level suspicious Flash file. A second-level suspicious Flash file that is hidden or small (i.e., *Metric 9* is true) is considered as a third-level suspicious Flash file.

3.2 Phase II: Prevention

In this phase, the Analyzer module is responsible for preventing fingerprinting attempts by changing the browser's fingerprint every time the user visits a website. For this purpose, we combine randomization (*Randomizer* in Fig. 1) and filtering (*Filter* in Fig. 1) techniques. Note that many websites need information about the browser (e.g., userAgent string, screen resolution, etc.) to customize their service for different browser or operating system configurations to run properly. As a result, we need a robust randomization technique which is able to: (i) change the browser's fingerprint between multiple visits and (ii) return values that represent the properties of the browser almost correctly. For example, the contents of a canvas element are pixels while the content of a `userAgent` property is a string representing the browser's name, version, and the underlying platform. Therefore, a suitable randomization technique is needed for each property. Moreover, our filtering technique should be able to remove fingerprinting

attempts (e.g., suspicious Flash files) for mitigation instead of disabling a plugin (e.g., the Flash plugin) for the whole webpage or the whole browser. In what follows, we describe the modules that we implemented in detail.

The Randomizer component implements four core engines: *objectRand*, *pluginRand*, *CanvasRand*, and *fontRand* to handle the respective fingerprinting attempts. The *objectRand* engine generates a random object at runtime to change the objects' properties between multiple visits. In this way, the objects' properties become an unreliable source of identity due to their randomized values. For this purpose, the Randomizer retrieves the `navigator` and `screen` objects of the browser, applies some changes on the properties shown to be important for fingerprinting (e.g., changing the subversion of the browser or adding noises to the current location of the user) [25], and replaces the generated objects with the native objects. However, for `plugins` and `mimeTypes` properties, this approach does not work, because changing the information of a browser's plugin might disable the plugin and cause loss of functionality. Therefore, the *pluginRand* engine adds a number of non-existing virtual `Plugin` and `MimeType` objects to the list of the current `plugins` and `mimeTypes` of the browser. It also changes the order of these lists upon every visit of the user to the URls in the blacklist.

Similarly, as canvas fingerprinting depends on the contents of canvas elements, the *CanvasRand* engine simply adds minor noises to the contents of a canvas element that is considered suspicious by the Analyzer component. Finally, the *fontRand* engine randomly reports the loaded available fonts as unavailable after the defined threshold has passed a limit. In this way, it changes the list of available fonts on the system and thus changes the fingerprint randomly. As a result, it assures that the webpage cannot employ JavaScript-based font detection for fingerprinting.

For combating Flash-based fingerprinting, instead of disabling the Flash plugin for the browser, we adopt two approaches (*flashFilter*): (i) filtering a Flash file that is identified as suspicious by the Analyzer component, and (ii) disabling the Flash plugin for each fingerprinter individually. In the former case, the Logger stores the URL of the suspicious Flash file in the blacklist. Then, the Analyzer watches for the URLs of the existing Flash files in the blacklist and prevents the browser from loading them. In the latter case, based on the Monitor (the component that flags the webpage as fingerprinter), the Logger stores the URL of the webpage in the blacklist. In subsequent visits, FPGuard disables the Flash plugin for the URL of the webpages that are present in the blacklist. For example, the Flash plugin can be disabled for a fingerprinter included as an `iframe` (which is considered as third-party) in another webpage. However, the Flash plugin can be enabled for the webpage that the user directly interacts with.

4 Implementation and Experimental Evaluation

In this section, we discuss the implementation of the FPGuard and its experimental evaluation in detail.

4.1 Implementation

FPGuard is developed as a combination of an instrumented version of the Chromium browser (version 38.0.2090.0) and a browser extension integrated with the Google Chrome browser. The FPGuard extension runs on the background of the browser and silently monitors and logs any activities related to fingerprinting on the browser. For example, to log the access to the `navigator`, `screen`, `Plugin`, and `MimeType` objects' properties, we override the getter method of these properties using the `Object.defineProperty` method. In order to record suspicious canvas elements, we override the methods of the `Object.prototype` of canvas elements that are used for writing (e.g., `fillText`) and retrieving the canvas contents (e.g., `getDataURL`). We add our implementation codes (in JavaScript) for recording the writes, reads, and the visibility status of the canvas elements (of the `Object.prototype` of canvas elements).

To record the suspicious Flash files that are loaded by the webpage, we first collect the URL of the Flash files that are loaded by the webpage. We use the *as3-commons* libraries [26] for parsing and decompiling Flash files. To do so, the FPGuard extension injects the decompiler's file (with the size of 132.4 KB) to the beginning of the webpage's DOM after the webpage is loaded. Next, the Logger sends the URL of Flash files to the decompiler. The decompiler decompiles the Flash files, traverses the obtained source codes for extracting the related metrics (see Sect. 3.1) and then sends back the obtained metrics for each URL to the Logger.

A fingerprinter might be able to check if FPGuard is installed or not by probing the getter method of the JavaScript objects' properties. However, the fingerprinter cannot invade FPGuard. Therefore, even if the fingerprinter finds out about the randomized attributes and skips them (because they are randomized and their actual value is not accessible), it generates a less unique fingerprint for the browser.

We should be clear that we were unable to find a way for having control on the browser fonts using JavaScript to prevent the JavaScript-based font detection method. The reason is that the style (i.e., font) is a property of each instance of JavaScript object not the prototype object. Thus it is not possible to filter a font on an HTML element before the loading of the font. Therefore, we modified the source code of a Chromium browser with the purpose of detecting and mitigating JavaScript-based font detection attempts. To this end, we identified the spots in the source code of the Chromium browser in which the browser loads the system fonts for HTML elements as well as the spots where the offset properties of HTML elements are returned upon being called through JavaScript. For this purpose, we modified the `CSSFontSelector.cpp` and `Element.cpp` classes. The `CSSFontSelector.cpp` class contains methods which are called upon the loading of the fonts for HTML elements (for prevention). The `Element.cpp` class contains methods for obtaining information about HTML elements such as offset properties (e.g.,width and height) using JavaScript (for detection).

Table 2. Presence of fingerprinting methods (i.e., JavaScript Objects (JSO), JavaScript-based Font Detection (JSFD), Canvas Fingerprinting (CF), and Flash-based Fingerprinting (FF)) in fingerprinters.

Fingerprinter	JSO	JSFD	CF	FF
bluecava	Yes	Yes	No	Yes
coinbase	Yes	Yes	No	No
browserleaks	No	No	Yes	No
fingerprintjs	Yes	No	Yes	No

4.2 Experimental Evaluation

Our evaluation of FPGuard is twofold: (i) measuring its effectiveness in correctly identifying fingerprinting-related activities at runtime, and (ii) measuring its effectiveness in protecting users from fingerprinters. With respect to the first part of our evaluation, similar to previous studies [6,7], we evaluate FPGuard using the top 10,000 websites from Alexa [27]. For the second part, whereas we use two types of fingerprinting providers in which the generated fingerprint is obtainable as an ID: a popular commercial fingerprinting provider named bluecava [12] and coinbase [28] and proof-of-concept fingerprinters named browserleaks [29] and fingerprintjs [30]. (see also Table 2)

Analysis of Detection. To automate the process of visiting the websites and data collection, we use the *iMacros* [31] extension for the Google Chrome browser. Using iMacros, we visit each website (of 10 K websites). Once the website is fully loaded, FPGuard records the metrics and identifies the activities of the website as either fingerprinting or normal. This way, we collected metrics for 9,264 websites (out of the 10 K websites). The remaining websites (736 websites) were not accessible at the time of our experiment. Next, we report the number of fingerprinting attempts that are discovered by FPGuard for each fingerprinting method.

JavaScript Objects fingerprinting. By using *Metric 1* and *Metric 2* (see Sect. 3.1), respectively, we enumerate the number of calls for the `navigator` and `screen` objects and the number of calls for the `Plugin` and `MimType` objects. The related thresholds for both metrics are defined accordingly. The maximum thresholds are when the websites look for all properties of the `navigator` and `screen` objects, and when they access a property for all plugins or MIME types, respectively. On average, the visited websites have looked up more than 5 properties of the `navigator` object and 3 properties of the `screen` object. In fact, more than 39 % of the visited websites have called the properties of the informative objects with more than the average (i.e., more than 5 properties of the `navigator` object and 3 properties of the `screen` object). As a result, returning invalid values for the properties of these objects might cause functionality loss of the browser as mentioned before. We also computed the suspiciousness level in performing fingerprinting based on the number of accesses to the properties of the informative

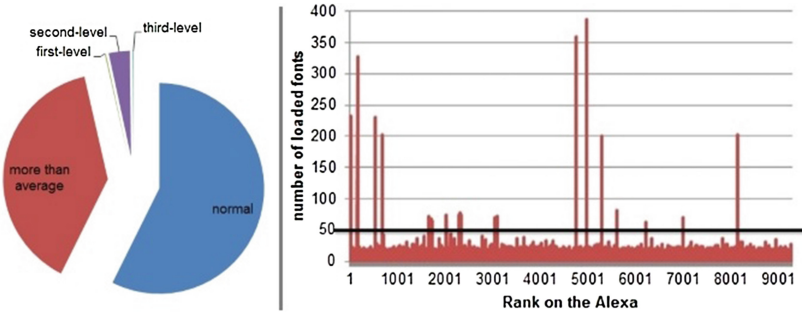

Fig. 2. Suspicious access level distribution for the Alexa's top 10 K websites (Left) and number of fonts that are loaded by the Alexa's top 10 K websites (Right).

objects for each individual website from our dataset. Figure 2 (left side) shows the result of measuring the suspiciousness level where most of the websites have normal (less than average) number of accesses to the properties of the informative objects (57.4 %). However, the number of websites that have suspicious access of first, second, and third level is considerably low in comparison to the number of websites that have the average number of accesses to the properties. Among the visited websites, only 101 of them requested for more than 80 % properties of the informative objects. That is to say, more than 17 properties of all 22 properties have been called. For example, Letitbit.net is one of the websites requested for most properties of the informative objects and the detailed information of all plugins and MIME types. This is due to, as we manually investigated, the fact that Letitbit contains a third-party script from MaxMind [32], which offers an online fraud prevention solution using fingerprinting. In addition, we visited bluecava and coinbase and for both, FPGuard reported second-level suspiciousness access to the informative objects' properties.

Finally, 15 websites from the dataset looked up for all the properties of the `navigator` and the `screen` objects plus the detailed information of all browser plugins and MIME types. Accessing all the information of `plugins` and `mimeTypes` is the characteristic of fingerprinting attempts. According to [25], using only these two properties, it is possible to identify a browser instance in a dataset of 1,523 browsers with almost 90 % accuracy.

JavaScript-based Font Detection. Based on our empirical studies, we revealed that most of the websites (92.76 % of 9,264 websites) request less than 50 fonts. Figure 2 (right side) shows the number of fonts that are loaded using JavaScript. Thus, we set 50 as the threshold for the number of fonts that a website can load. In addition, according to our analysis, the threshold for the average number of accesses to the offset properties of HTML elements is set to 65. Therefore, a given website that loads more than 50 fonts and the number of its accesses to the offset properties of HTML elements surpasses 65 is considered as a fingerprinting candidate. FPGuard identified 22 websites that were loading more than 50 fonts and accessed more than 69 times to the offset properties of HTML elements. Among the 22

websites that loaded more than 300 different fonts with more than 69 accesses to the offset properties of HTML elements are: www.yad2.co.il (388), www.junkmail.co.za (359), www.vube.com (327), and www.people.com.cn (325). These websites either employ fingerprinting themselves or contain a third-party script from a fingerprinting provider.

Canvas Fingerprinting. Figure 3 (left side) shows the distribution of suspicious (first and second-level) canvas elements among the visited websites. FPGuard found 191 websites from our dataset with first and second-level suspicious canvas elements. However, by analyzing their image data, only 85 out of 191 are second-level suspicious elements performing canvas fingerprinting. As shown in Fig. 3 (right side), in all cases, the fingerprinter inserts an arbitrary text, often a pangram with different colors to increase the uniqueness of the fingerprint.

However, a recent study identified canvas fingerprinting as the most prevalent type of fingerprinting among the top 10 K websites of Alexa (4.93 %) [7]. The study found that 95 % of the canvas fingerprinting attempts come from a third-party script which belongs to a single fingerprinting provider (i.e., AddThis [33]). In fact, after manual investigation, we found that AddThis disabled canvas fingerprinting as of mid July, 2014 [34]. Websites that use canvas fingerprinting belong to categories such as dating (e.g., www.pof.com), news (e.g., www.reuters.com), file sharing (e.g., www.letitbit.com), deal-of-the-day (e.g., www.groupon.com), etc. These websites either include a third-party script whose main job is fingerprinting (e.g., www.reuters.com and www.letitbit.com) or they actually perform fingerprinting (e.g., pof.com and groupon.com). For example, reuters.com includes a script from Audience Science [35] which is an advertising company (from revsci.net domain) that implements canvas fingerprinting. This company uses Fingerprintjs [30], which implements canvas fingerprinting.

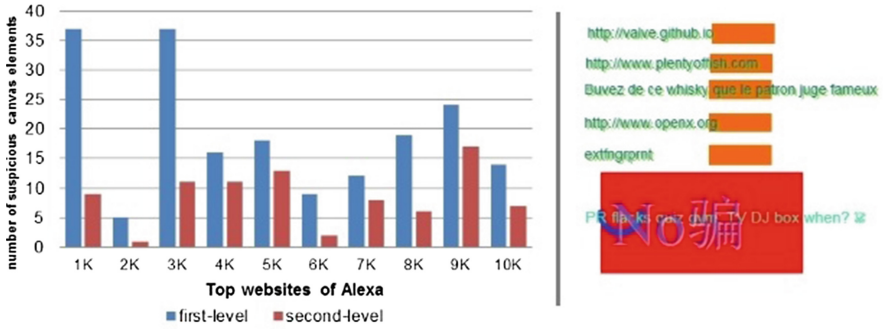

Fig. 3. Number of suspicious canvas elements that are found from our dataset (Left) and different canvas images used for fingerprinting browsers (Right).

Flash-based Fingerprinting. FPGuard found 124 websites which included Flash files in their homepage that contain methods for enumerating system fonts

and considered them as first-level suspicious. We revealed that 120 out of the 124 websites loaded Flash files that include methods for storing information on the user's system, sending information to a remote server, or interacting with JavaScript programs. Hence, they are marked as second-level suspicious Flash files. From the 120 websites, only 59 of them contain third-level suspicious Flash files, meaning that the Flash files are hidden, small (less than 2×2 pixels), or added dynamically. We manually investigated the source code of the third-level suspicious Flash files and observed that they indeed perform fingerprinting. In addition, four second-level suspicious Flash files are observed to perform fingerprinting.

Discussion on FPGuard's Detection Accuracy. As for detection accuracy of FPGuard, we attempted to calculate its false-positive and false-negative rates. A false-negative error occurs when FPGuard improperly identifies a webpage as normal while the webpage is fingerprinter. On the other hand, a false-positive error occurs when FPGuard identifies a webpage as fingerprinter while the webpage is normal. Regarding the false-negative rate, FPGuard identified all fingerprinting websites (known as fingerprinter) as fingerprinters (false-negative rate is 0 %). However, for the false- positive rate, we find that it is challenging to define this value for fingerprinting-related topics. The reason is that fingerprinting is just collecting data from the users' browsers. That is to say, unless we know the name of the company (and identify it as tracker), we cannot say they are performing fingerprinting. However, we observed that many websites in our database are performing fingerprinting-related activities while they used their own scripts (mostly obfuscated), and they did not have a script from a known tracker. As a result, we cannot label them as fingerpriner; instead, we consider these websites as candidates for performing fingerprinting attempts. Accordingly, it is difficult if not impossible to evaluate the false-positive rate in our database.

Analysis of Prevention. To assess the accuracy of FPGuard's prevention technique, we repeatedly visited each fingerprinter for 100 consecutive times and collected the generated IDs. Then, we checked the uniqueness of the generated IDs with the goal of achieving the highest unique value from the list of IDs. In other words, a single browser can have 100 different IDs when one prevention technique is enabled; and, thus, the fingerprinter cannot uniquely identify the browser upon multiple visits. We define uniqueness as the fraction of the number of unique IDs and the total number of IDs. We found that the fingerprinters generate a unique ID for the browser upon each visit when FPGuard is enabled (i.e., 100 different fingerprints for a given browser upon each visit when visiting each fingerprinter for 100 times). This means that the fingerprinter is not able to uniquely identify the browser between multiple visits.

To evaluate the capability of the two popular commercial fingerprinters (i.e., bluecava and coinbase), we slightly modified some browser information (e.g., version of the browser) to change the browser's fingerprint in order to check whether the fingerprinter can detect the changes and thus generate an identical ID for the same browser. Surprisingly, none of the fingerprinting providers could detect

slight changes made to the fingerprint. For example, bluecava generated different IDs for a browser instance upon every visit when the order of the elements in the browser's plugin list was changed randomly. Moreover, Bluecava could not detect the changes in the browser's version and generated different IDs when changes were performed on the browser's version. Similarly, we evaluated browserleaks and fingerprintjs by adding a single-pixel random noise to the content of canvas elements and checked whether they generate an identical ID for the browser or not. They both generated new IDs, demonstrating that they highly depend on the canvas contents and cannot detect minor changes. Our analysis of the four fingerprinters showed that the current fingerprinting algorithms are not capable of detecting slight changes of a fingerprint. In addition, gradual changes such as updating the browser version cannot be detected by even popular fingerprinters. The main point here is that our randomization and filtering engines are used to make the changes even undetectable to prevent fingerprinting.

5 Comparative Discussion

As mentioned earlier, FPdetective browser extension is a blacklist-based approach for detecting previously discovered fingerprinting objects. To compare FPGuard with FPdetective, we first included the blacklisted URLs that exist in the FPdetective extension in FPGuard. After visiting each website from our dataset, FPGuard identified 36 websites that contain URLs from the blacklist. This means that these 36 websites contain methods for fingerprinting (i.e., using *JavaScript-based Font Detection* and *Flash-based Fingerprinting* methods) and are detected by FPdetective. FPdetective identified 9 websites (out of 36 websites) as fingerprinter but we did not find any fingerprinting-related activities within these websites. We manually investigated the URLs that are identified by FPdetective as sources of fingerprinting and did not find any fingerprinting attempts which match with fingerprinting patterns of *JavaScript-based Font Detection* and *Flash-based Fingerprinting* methods. Furthermore, we observed that each of the identified websites (i.e., the 9 websites) include a Flash file[1,2] which is identified as suspicious by FPdetective. However, our analysis of these files showed that they contain methods for storing evercookies [36] or Flash cookies on the user's system (i.e., there are neither any attempts for collecting system-related information nor for enumerating system fonts). Finally, compared to FPGuard, the FPdetective extension cannot detect newly added fingerprinting scripts or Flash objects due to its blacklist-based nature.

From the prevention point of view, Tor and Firegloves block both normal and fingerprinting websites as they are not equipped with a detection mechanism. However, FPGuard first detects fingerprinting attempts and then applies a suitable prevention technique. Moreover, as mentioned earlier, FPGuard returns neither fixed nor empty values for the browser properties. Conversely, Tor and

[1] http://bbcdn-bbnaut.ibillboard.com/server-static-files/bbnaut.swf.
[2] https://aa.online-metrix.net/fpc.swf.

Firegloves return empty values for the plugin list. In our experiments, a property of the browser's plugin list is looked up by nearly 75.2 % of Alexa's top 10 K websites. Thus, returning an empty value for the plugins list can cause functionality loss. While Tor disables all plugins in the browser, PriVaricator employs a randomization-based solution by randomly hiding some plugins from the browser's plugin list upon every visit of a user to a webpage. Hiding a number of plugins still causes functionality loss of the browser. In contrast, FPGuard adds non-existing plugins with real names and properties to the browser's plugins list and permutes the order of the plugins upon every visit.

In addition, Tor, Firegloves and ExtensionCanvasFingerprintBlock return empty contents for all canvas elements. However, FPGuard keeps the browser's functionality by first identifying suspicious canvas elements and then adding minor noises to their contents. For preventing *JavaScript-based Font Detection* attempts, both Tor and Firegloves limit the number of system fonts for all websites. PriVaricator adds random noises to the offset properties of HTML elements after they are looked up more than 50 times. As mentioned earlier, this approach might interfere with the design of certain websites. FPGuard, on the other hand, first detects *JavaScript-based Font Detection* attempts and then after a predefined threshold (50 fonts), it randomly announces the available fonts as unavailable. In this way, it assures the randomness of browser's fingerprint on multiple visits. Moreover, it does not interfere with the webpage's design since it has no effect on the offset properties of the HTML elements and it only randomly changes the available fonts after a predefined threshold.

6 Conclusion

In this paper, we have presented the design and implementation of FPGuard — an approach to detect and combat fingerprinters at runtime. For detection of browser fingerprinting, we used a number of algorithms to analyze the metrics related to each fingerprinting method. In the detection phase, we set the threshold values based on our definition of abnormal behavior. For example, we selected 50 fonts as the threshold for JavaScript-based font detection, because we found that top websites (99.7 % of top 10 K websites) asked for less than 50 fonts. As for fingerprinting prevention, we combined randomization and filtering techniques to change the browser fingerprint in every visit the user makes to a webpage. In particular, we applied suitable randomization policies (4 policies) and filtering techniques to keep the browser functionality and combat fingerprinting attempts.

We evaluated FPGuard's detection approach on Alexa's top 10,000 websites and identified *Canvas Fingerprinting* as the most prevalent type of fingerprinting method on the Web (85 websites out of 10,000 websites). In addition, we evaluated FPGuard against a blacklist-based fingerprinting detector (i.e., FPdetective) and showed that FPGuard outperformed FPdetective's capability in correctly identifying the fingerprinters. More specifically, FPdetective identified 9 websites as fingerprinters exploiting the *Flash-based Fingerprinting* method, but we could not find any fingerprinting-related attempts in the source code of the Flash files that were identified as evidences of fingerprinting.

We also evaluated FPGuard's prevention approach against four fingerprinting providers. FPGuard successfully (with a negligible overhead) changed the browser's fingerprint upon each visit to the fingerprinters. This is to say, the fingerprinters were not able to uniquely identify the browser between multiple visits. We also compared FPGuard with the existing anti-fingerprinting tools and pointed out the strengths and weaknesses of each tool. We observed that the existing solutions cause functionality loss of the browser for both normal and fingerprinter webpages. However, FPGuard wins over fingerprinting attempts by keeping the browser functionality for the normal websites. It does by first detecting fingerprinting attempts and then preventing them using randomization policies and filtering techniques. Finally, we analyzed the existing fingerprinting algorithms used in popular fingerprinting providers.

References

1. Boda, K., Földes, Á.M., Gulyás, G.G., Imre, S.: User tracking on the web via cross-browser fingerprinting. In: Laud, P. (ed.) NordSec 2011. LNCS, vol. 7161, pp. 31–46. Springer, Heidelberg (2012)
2. Eckersley, P.: How unique is your web browser? In: Atallah, M.J., Hopper, N.J. (eds.) PETS 2010. LNCS, vol. 6205, pp. 1–18. Springer, Heidelberg (2010)
3. Wondracek, G., Holz, T., Kirda, E., Kruegel, C.: A practical attack to de-anonymize social network users. In: Proceedings of the 2010 IEEE Symposium on Security and Privacy, SP 2010, pp. 223–238. IEEE Computer Society, Washington, DC, USA (2010)
4. Ghostery. Ghostery — home (2014). https://www.ghostery.com/en/
5. AdblockPlus. Adblock plus — home (2014). https://adblockplus.org/
6. Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., Preneel, B.: Fpdetective: dusting the web for fingerprinters. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & #38; Communications Security, CCS 2013, pp. 1129–1140. ACM, New York, NY, USA (2013)
7. Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., Diaz, C.: The web never forgets: persistent tracking mechanisms in the wild. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS 2014, pp. 674–689. ACM, New York, NY, USA (2014)
8. Hoofnagle, C.J., Urban, J.M., Li, S.: Privacy and modern advertising: most us internet users want'do not track'to stop collection of data about their online activities. In: Amsterdam Privacy Conference (2012)
9. McDonald, A.M., Cranor, L.F.: Beliefs and behaviors: internet users? understanding of behavioral advertising. In: Proceedings of the 2010 Research Conference on Communication, Information and Internet Policy. Carnegie Mellon University, Pittsburgh (2010)
10. Turow, J., King, J., Hoofnagle, C.J., Bleakley, A., Hennessy, M.: Americans reject tailored advertising and three activities that enable it (2009). (SSRN 1478214)
11. TRUSTe. 2009 study: Consumer attitudes about behavioral targeting (2009). http://goo.gl/LO2QEm
12. Bluecava. Bluecava — opt-out (2014). http://bluecava.com/opt-out
13. AddThis. Addthis - terms of service, July 2014. http://www.addthis.com/tos

14. Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., Vigna, G.: Cookieless monster: exploring the ecosystem of web-based device fingerprinting. In: Proceedings of the 2013 IEEE Symposium on Security and Privacy, SP 2013, pp. 541–555. IEEE Computer Society, Washington, DC, USA (2013)
15. Perry, M., Clark, E., Murdoch, S.: The design and implementation of the tor browser[draft], July 2014. https://www.torproject.org/projects/torbrowser/design/
16. Nikiforakis, N., Joosen, W., Livshits, B.: Privaricator: Deceiving Fingerprinters with Little White Lies (2014). http://research.microsoft.com/apps/pubs/default.aspx?id=209989
17. Boda, K.: Firegloves (2014). http://fingerprint.pet-portal.eu/?menu=6
18. Mayer, J.R.: Any person... a aamphleteer?: Internet anonymity in the age of web 2.0. Undergraduate Senior Thesis, Princeton University (2009)
19. Mowery, K., Bogenreif, D., Yilek, S., Shacham, H.: Fingerprinting information in javascript implementations. In: Proceedings of W2SP (2011)
20. Olejnik, L., Castelluccia, C., Janc, A., et al.: Why johnny can't browse in peace: on the uniqueness of web browsing history patterns. In: 5th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2012) (2012)
21. Mulazzani, M., Reschl, P., Huber, M., Leithner, M., Schrittwieser, S., Weippl, E., Wien, F.H.C.: Fast and reliable browser identification with javascript engine fingerprinting. In: Web 2.0 Workshop on Security and Privacy (W2SP), vol. 5 (2013)
22. Yen, T.-F., Xie, Y., Fang, Y., Yu, R.P., Abadi, M.: Privacy and security implications. In: NDSS, Host fingerprinting and tracking on the web (2012)
23. Murdoch, S., Perry, M., Clark, E.: Tor: Cross-origin fingerprinting unlinkability (2014). https://www.torproject.org/projects/torbrowser/design/#fingerprinting-linkability
24. Appodrome. Canvasfingerprintblock (2014). http://goo.gl/1ltNs4
25. Khademi, A.F., Zulkernine, M., Weldemariam, K.: Empirical evaluation of web-based fingerprinting (to appear in ieee software, 2015). IEEE Software's SWSI: Security & Privacy on the Web (2015)
26. as3commons. As3 commons bytecode (2014). http://www.as3commons.org/as3-commons-bytecode/index.html
27. Alexa. Alexa - actionable analytics for the web (2014). http://www.alexa.com/
28. CoinBase. Coinbase - an international digital wallet (2014). https://coinbase.com/
29. BrowserLeaks.com. Browserleaks (2014). https://www.browserleaks.com/
30. Valve. Fingerprintjs (2014). https://github.com/Valve/fingerprintjs
31. imacros. imacros for chrome (2014). https://chrome.google.com/webstore/detail/imacros-for-chrome/cplklnmnlbnpmjogncfgfijoopmnlemp?hl=en
32. MaxMind. Maxmind - ip geolocation and online fraud prevention (2014). https://www.maxmind.com/en/home
33. AddThis. Addthis - get likes, get shares, get followers (2014). http://www.addthis.com/privacy/opt-out
34. AddThis. The facts about our use of a canvas element in our recent r&d test (2014). http://goo.gl/qa0x1y
35. AudienceScience. Audiencescience - enterprise advertising management system (2014). http://www.audiencescience.com/
36. Kamkar, S.: Evercookie-never forget. New York Times (2010)