FPDetective: Dusting the Web for Fingerprinters

Gunes Acar[1], Marc Juarez[1,2], Nick Nikiforakis[3], Claudia Diaz[1], Seda Gürses[1,4], Frank Piessens[3] and Bart Preneel[1]

[1]KU Leuven, Dept. of Electrical Engineering (ESAT), COSIC, iMinds, Leuven, Belgium
{gunes.acar,marc.juarez,claudia.diaz,seda.guerses,bart.preneel}@esat.kuleuven.be

[2]IIIA-CSIC, Bellaterra, Spain
mjuarez@iiia.csic.es

[3]KU Leuven, Dept. of Computer Science, iMinds-DistriNet, Leuven, Belgium
{nick.nikiforakis,frank.piessens}@cs.kuleuven.be

[4]New York University, Dept. of Media, Culture, and Communication, NY, USA

ABSTRACT

In the modern web, the browser has emerged as the vehicle of choice, which users are to trust, customize, and use, to access a wealth of information and online services. However, recent studies show that the browser can also be used to invisibly fingerprint the user: a practice that may have serious privacy and security implications.

In this paper, we report on the design, implementation and deployment of FPDetective, a framework for the detection and analysis of web-based fingerprinters. Instead of relying on information about known fingerprinters or third-party-tracking blacklists, FPDetective focuses on the detection of the fingerprinting itself. By applying our framework with a focus on font detection practices, we were able to conduct a large scale analysis of the million most popular websites of the Internet, and discovered that the adoption of fingerprinting is much higher than previous studies had estimated. Moreover, we analyze two countermeasures that have been proposed to defend against fingerprinting and find weaknesses in them that might be exploited to bypass their protection. Finally, based on our findings, we discuss the current understanding of fingerprinting and how it is related to Personally Identifiable Information, showing that there needs to be a change in the way users, companies and legislators engage with fingerprinting.

Categories and Subject Descriptors

K.6.m [**Management of Computing and Information Systems**]: Miscellaneous; H.3.5 [**Information Storage and Retrieval**]: Online Information Services — *Web-based services*; K.4.4 [**Computers and Society**]: Electronic Commerce — *Security*

Keywords

Web security; privacy; device fingerprinting; tracking; dynamic analysis; JavaScript; Flash

CCS'13, November 4–8, 2013, Berlin, Germany.

http://dx.doi.org/10.1145/2508859.2516674.

1. INTRODUCTION

In 2010, Eckersley demonstrated that benign characteristics of a browser's environment, like the screen dimensions and list of installed fonts, could be combined to create a unique device-specific fingerprint [7]. Of the half million users who participated in Eckersley's experiment, 94.2% of those using Flash or Java had a unique device fingerprint, and could thus be identified and tracked without the need for stateful client-side technologies, such as browser or Flash cookies.

Fingerprinting user devices through the browser is an increasingly common practice used of advertising and anti-fraud companies. Stateless user tracking allows advertising companies to sidestep the limitations imposed by regulation on *cookies* in Europe and the United States. Moreover, with the advent of smartphones and tablets, fingerprinting allows advertisers to augment previously gathered user-data and track the user across devices. Anti-fraud companies advertise fingerprinting as a means to protect users and web applications against malevolent actors, for instance, by detecting the use of stolen credentials and identifying Sybil attacks [6]. Most of the time, these services are based on massive device reputation databases where device fingerprints are stored along with the device owners' web history and "reputation scores."

Device fingerprinting raises serious privacy concerns for everyday users. Its stateless nature makes it hard to detect (no cookies to inspect and delete) and even harder to opt-out. Moreover, fingerprinting works just as well in the "private-mode" of modern browsers, which cookie-conscious users may be utilizing to perform privacy-sensitive operations. In recent research, Nikiforakis et al. [18] analyzed the techniques and adoption of three fingerprinting companies,

two of which had been identified by Mayer and Mitchell in a 2012 survey paper on web-tracking [13]. While the authors studied the practices of these three companies, they did not attempt to discover other fingerprinters and explore the methods adopted by these.

This paper aims to shed light on current device fingerprinting practices through three main contributions. The first is the design and implementation of *FPDetective*,[1] a framework for identifying and analyzing web-based device fingerprinting without relying on a list of known fingerprinters.

Second, we use FPDetective to conduct a large-scale study of web-based device fingerprinting in the top million Alexa sites. Using FPDetective, we were able to identify 16 new fingerprinting scripts and Flash objects (including commercial fingerprinting as well as in-house solutions), some of which are active in the top 500 websites, showing that fingerprinting is much more prevalent than previous studies estimated. Further, we uncovered previously unreported fingerprinting practices, such as attempting to evade detection by removing the fingerprinting script once the device has been fingerprinted, and collecting fingerprints through third-party widgets. Our findings highlight the rising prevalence of fingerprinting and, in turn, the need for more transparency, awareness and counter-measures with respect to these practices.

Finally, we evaluate the Tor Browser and Firegloves, two privacy-enhancing tools that have "fingerprinting-resistance" as a design goal. We discovered vulnerabilities in both tools that would allow a fingerprinter to identify users. This illustrates the difficulty of protecting against device fingerprinting. We also tested whether enabling the Do-Not-Track (DNT) header had any impact on the behavior of fingerprinting scripts, and found that it does not.

The rest of the paper is organized as follows. Section 2 reviews the state-of-the-art in device fingerprinting methods. The FPDetective framework is described in Section 3. Section 4 motivates the use of font detection to identify potential fingerprinters, and explains our data analysis methodology. Section 5 presents the experimental results of our study of the top million Alexa websites. We evaluate in Section 6 the Tor browser, Firegloves, and DNT as countermeasures to fingerprinting. We discuss in Section 7 the uses of fingerprinting and its relationship to privacy concerns. Finally, we offer our conclusions in Section 8.

2. DEVICE FINGERPRINTING

A device *fingerprint* is a set of system attributes that, for each device, take a combination of values that is, with high likelihood, unique, and can thus function as a device identifier. These attributes include, for example, the device's screen size, the versions of installed software, and the list of installed fonts. Attributes that take more diverse values (e.g., the list of fonts) are more identifying than values shared by many devices (e.g., version of the operating system). Similarly, attributes with values that are more stable over time (i.e., that change only infrequently or very gradually) facilitate identification compared to those that change often and unpredictably.

Web-based device fingerprinting is the process of collecting sufficient information through the browser to perform stateless device identification. These fingerprints may then be used as identifiers for tracking the device in the web. By tracking, we refer to the linking of visits to (one or multiple) web pages as made by the same device.

There are many reasons why web applications may need device-related information, e.g., to correctly render content or to serve device compatible media. Thus, there are many APIs that enable applications to query these attributes. At the same time, these APIs can be used to learn enough attribute values to obtain a device fingerprint that is, for practical purposes, unique. When a user browses to a webpage that includes fingerprinting software, her device fingerprint may be collected and compared to a database of known devices. Using such techniques, known devices can be matched, and previously unknown devices can be added to the database. Depending on the use case, the database entry for each device is augmented with contextual and behavioral information with each visit of the user to a monitored webpage.

In this section we briefly introduce device fingerprinting methods. Some of the discussed methods have already been encountered in real-life fingerprinting code, while others have been shown to work but are not yet known to have been adopted by fingerprinters [18].

2.1 JavaScript-based

From its original inception in 1995 all the way to today, JavaScript has emerged as the de-facto client-side, programming language of the web. The expressiveness of JavaScript, combined with its "forgiving" nature when it comes to programming errors (such as missing semicolons) and, most importantly, its ubiquitous availability in modern browsers, has made it an indispensable tool of modern web sites. Virtually all non-static web pages use JavaScript and a 2010 study by Yahoo showed that only 1% of actual human visitors have JavaScript disabled [24].

JavaScript is used by programmers mainly to dynamically manipulate a page's DOM, enrich user experience through asynchronous requests and responses, and offload non-critical, server functionality to the client. Its privileged position inside the browser, however, also makes it a strong fingerprinting tool. The JavaScript-accessible browser resources that have, historically, been probed the most, are the following:

- `navigator`: The `navigator` object contains information about the browser vendor and specific browser version, the supported plugins and MIME types, as well as relatively coarse-grained information about the operating system and architecture on which the browser is executing.
- `screen`: The `screen` object contains information about the resolution of the user's monitor (height and width) and the color and pixel depth.

Mayer, in 2009, reported on an experiment where he fingerprinted 1328 web clients [12]. By hashing the concatenated contents of the `navigator`, `screen`, `navigator.plugins` and `navigator.mimeTypes`, Mayer was able to uniquely identify more than 96% of the browsers. A year later Eckersley, through the Panopticlick project[2], fin-

[1]The FPDetective framework is available here: `http://homes.esat.kuleuven.be/~gacar/fpdetective/`

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

gerprinted nearly half a million browsers and by extending the set of fingerprinted features with fonts, timezones and a browser's ACCEPT headers, was able to uniquely identify 94.2% of the visitors' browsing environments [7]. Eckersley also showed that the list of installed fonts is one of the most identifying features of a system. This list can be obtained either through JavaScript by measuring and then comparing the dimensions of text rendered with different fonts [18], or through browser plugins.

Other researchers have proposed the use of performance benchmarks for differentiating between JavaScript engines [15], errors in standard test-suites [17] and differences in the appearance of `canvas` elements created through JavaScript [16]. Furthermore, a user's browsing history, which can be recovered exploiting JavaScript's visited-link color feature [9], has also been shown to uniquely identify users [19]. In general, while these methods have not yet been encountered in deployed fingerprinting products [18], they could potentially be used to increase the accuracy of the gathered fingerprints.

2.2 Plugin-based

The latest version of HTML, HTML5, together with the advanced capabilities of JavaScript and Cascading Style Sheets, give web developers today the ability to create feature-rich web applications. This, however, was not the case with older versions of HTML, as their abilities to deliver interactive rich Internet applications, like games, video and music were limited. Third-party companies, like Adobe, developed plugins and platforms to create, deliver and render interactive multimedia content. The clear winner of this "plugin-war" was Adobe Flash, with Java being a distant second.

As with JavaScript, the adoption of popular third-party plugins gives fingerprinters the ability to extract numerous features. Eckersley used Java and Flash to obtain the list of fonts installed in a device [7], since font-enumeration API calls are made available by the Flash and Java plugins. In addition to font extraction, commercial fingerprinting companies use Flash to circumvent HTTP proxies set up by the user and get more fine-grained information about the device, such as the specific operating system kernel version, or the presence of multiple-monitor setups [18].

2.3 Extension-based

The modular nature of modern browsers has allowed developers to create extensions that add new functionality, remove undesired features, and modify others. Unlike plugins, extensions are not enumerable through JavaScript and thus can only be detected by their possible side-effects. For instance, Mowery et al. [15] showed that it is possible to deduce custom whitelists from the popular NoScript plugin, simply by requesting scripts from domains and later inspecting whether the scripts successfully executed, by searching for predetermined JavaScript objects in the global address space. The deduced whitelists can be used as an extra fingerprint feature. Nikiforakis et al. [18] showed that user-agent-spoofing extensions can also be discovered due to inconsistencies in the reported browsing environment when each extension is active.

2.4 Header-based & Server-side

Yen et al. [23] performed a fingerprinting study, similar to Eckersley's, by analyzing month-long logs of Bing and Hotmail. The authors found that IP addresses and user-agent headers may help in tracking users with high precision and recall.

Predating fingerprinting at the browser-level, researchers had shown that it is possible to not only remotely learn the operating system of a particular host on the Internet [1, 25], but also to fingerprint multiple physical devices hidden behind NATs through their clockskew, by analyzing the TCP timestamps of network packets [11].

While these techniques may be less accurate compared to the aforementioned in-browser fingerprinting methods, their pure server-side nature makes their detection very difficult, if not impossible.

3. FPDETECTIVE FRAMEWORK

In this section we describe FPDetective, a framework for detecting web-based device fingerprinting. FPDetective is designed as a flexible, general purpose framework that can be used to conduct further web privacy studies. FPDetective is freely available and can be downloaded from `http://homes.esat.kuleuven.be/~gacar/fpdetective`.

Figure 1 outlines FPDetective's components and workflow. The main component of FPDetective is a crawler, whose purpose is to visit websites and collect data about events that might be related to fingerprinting, such as the loading of fonts, or accessing specific browser properties. These logs are parsed and committed to central database in a relational structure. In order to detect Flash-based fingerprinting, all browser traffic is directed through an intercepting proxy that logs all the HTTP(S) traffic between the browser and the web server. These network dumps are parsed to extract Flash objects, that are then decompiled using a free, third-party decompiler and stored in the database.

In our analysis of the data collected by FPDetective we focus on font detection. We would like to emphasize that this choice was made in order to facilitate the analysis of the data, and the data gathered by FPDetective can be analyzed differently, based on other fingerprinting classification criteria. Further, the FPDetective framework can easily be adapted for use in Web privacy studies unrelated to fingerprinting. The framework is developed with modularity in mind using Python, C++, JavaScript and MySQL programming/scripting languages. Researchers can customize the framework to carry out different experiments by replacing the script that FPDetective executes when it visits the sites. We warmly welcome other researchers to provide their comments, contribute to the project, or fork their own software out of FPDetective.

The remainder of this section describes each component of FPDetective in more detail.

3.1 Components

Crawler: The crawler features two instrumented browsers, PhantomJS[3] and Chromium[4]. We chose PhantomJS to collect data related to JavaScript-based fingerprinting for its minimal use of resources. We used Chromium to investigate Flash-based fingerprinting, since PhantomJS does not have plugin support and thus cannot run Flash objects.

[3] `http://phantomjs.org/`
[4] `http://www.chromium.org/`

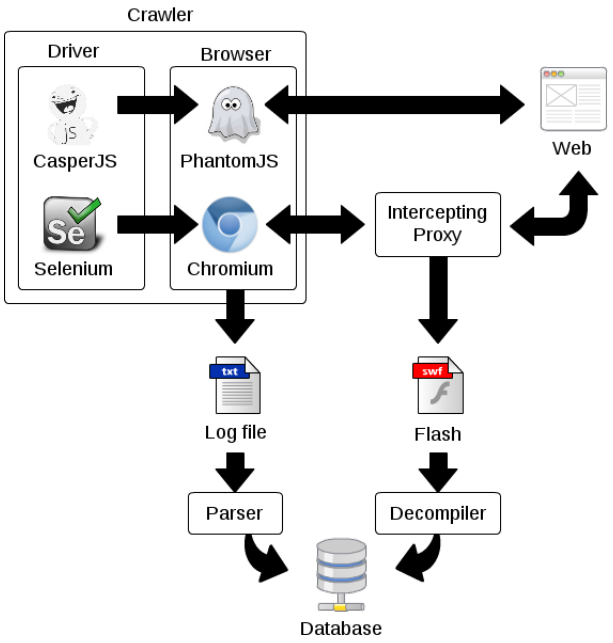

Figure 1: FPDetective Framework

CasperJS[5] and Selenium[6] were used to drive the browsers to websites and navigate through the pages.

To build instrumented versions of the browsers, we modified parts of the WebKit source code, which is the rendering engine used by both Chromium and PhantomJS. It should be noted, however, that during the course of our study, Chromium Project announced that they leave WebKit for a new rendering engine called Blink, which is again based on WebKit [2]. We preferred to work at the native code level instead of developing browser extensions or JavaScript patches for several reasons: to detect events that are not JavaScript-based (especially those related to fonts); to detect the origin of events more precisely; and to defend against JavaScript attacks that block or circumvent extensions and getter methods[7].

The modifications allow the crawler to intercept and log accesses to the following browser and device properties, which could be used for fingerprinting:

- access to the following `navigator` properties and methods: `userAgent`, `appCodeName`, `product`, `productSub`, `vendor`, `vendorSub`, `onLine`, `appVersion`, `language`, `plugins`, `mimeTypes`, `cookieEnabled()`, `javaEnabled()`
- access to `navigator.plugins`: `name`, `filename`, `description`, `length`
- access to `navigator.mimeTypes`: `enabledPlugin`, `description`, `suffixes`, `type`
- access to `window.screen` properties: `horizontalDPI`, `verticalDPI`, `height`, `width`, `colorDepth`, `pixelDepth`, `availLeft`, `availTop`, `availHeight`, `availWidth`
- access to `offsetWidth` and `offsetHeight` properties and `getBoundingRect` method of HTML elements.
- font load attempts by intercepting calls to `CSSFontFace::getFontData` and `CSSFontSelector::getFontData` methods

Parser: The parser is used to extract relevant data from the logs generated by the crawler, and to store them in the database. It also tags sites with a label if a known fingerprinting script is found in the HTTP requests made for this visit.

Intercepting Proxy: In order to obtain Flash files for static analysis, we redirected traffic through mitmproxy [4], an SSL-capable intercepting proxy. We used the mitmdump module to log all the HTTP traffic passing through the proxy, and the libmproxy library to parse and extract Flash files based on content sniffing. More specifically, we detected Flash files through Flash-specific "magic numbers" appearing the first bytes of content since the `Content-Type` HTTP header is not always reliable.

Decompiler: We used the JPEXS Free Flash Decompiler[8] to decompile Flash files and obtain the ActionScript source code. The source code is then searched for fingerprinting related function calls (e.g. `enumerateFonts` and `getFontList`) to obtain a binary occurrence vector. The complete list of methods and properties searched in the decompiled source code is available in Appendix B.

Central Database: We ran crawls using several machines, but used a central database to store, combine, and analyze the results of different crawls with minimal effort. The stored data include the set of JavaScript function calls, the list of HTTP requests and responses, and the list of loaded or requested fonts. For the Flash experiments, we also stored a binary vector that represents the existence of ActionScript API calls that might be related to fingerprinting.

3.2 Performance

By using the Dromaeo JavaScript performance test suite[9] we compared our modified Chromium browser against the Chromium browser available from Ubuntu's repositories. Although the versions of Chromium and WebKit were different for the two browsers (25 vs 28 and 537.22 vs 537.36 repectively) we still believe that the results can provide a rough approximation of the performance of our tool. The resulting scores from the Dromaeo test suites were 3,148.63 runs/s for original Chromium and 3,025.86 runs/s for our modified browser. The difference between their aggregate performance is about 4%, where error intervals were measured as 1.24% and 1.98%.

The most crucial differences were between the `getHeight` and `getWidth` methods of the Prototype JavaScript library[10] (9.76 vs 214.1 and 9.79 vs 244.3) which is probably due to the fact that we log accesses to `offsetHeight` and `offsetWidth` properties. But within JQuery's[11] `height` and `width` methods, differences were quite low (0.84 vs 2.27 and 0.97 vs

[5] http://casperjs.org/
[6] http://docs.seleniumhq.org/
[7] http://code.google.com/p/chromium/issues/detail?id=55084
[8] http://www.free-decompiler.com/flash/
[9] http://dromaeo.com/
[10] http://prototypejs.org/
[11] https://jquery.org/

2.27) hinting that this might also be due to the libraries' implementation details[12].

We were able to run 200 parallel PhantomJS instances on a customer grade computer with quad core CPU and 8GB of RAM. With these settings we were able to complete a homepage crawl of the Alexa top million sites in a period of four days.

For the experiments with the Chromium browser, we could run approximately 20 crawlers in parallel, since each crawler was running an instance of Chromium and mitmproxy, as well as was using the Flash decompiler whenever a Flash object was received. Crawling the top 20,000 Alexa sites required about one day using a single computer with the characteristics given above.

4. FONT-BASED ANALYSIS OF FINGERPRINTING

In order to carry out a large scale study of fingerprinting, we combined automated and manual analysis in our experiments. In the automated step we use font probing and font enumeration as a criteria for identifying candidate sites that possibly include a fingerprinting script or object. This approach is motivated in Section 4.1 and explained in more detail in Section 4.2. We then examine manually the candidates to refine the classification using additional criteria and establish whether the pages contain fingerprinting code.

4.1 Motivation

By inspecting the code of known fingerprinters, and taking into account the findings of previous studies [7, 18], we distill *"font detection"* through enumeration or probing as a necessary ingredient of device fingerprinting. Furthermore, we consider font detection to be a good indicator of fingerprinting, as getting the list of system fonts has fewer use cases than requesting information needed for browser feature detection.

The use of font detection for fingerprinting also offers the following advantages:

- according to Panopticlick study, fonts are the second most identifying attribute of a device, having 17.1 bits of entropy when unsorted;
- although the list of installed browser plugins has slightly more entropy, fonts are OS-dependent and thus enable linking different browsers running on the same device;
- the Tor Browser, which is the most widely used tool with counter-fingerprinting features, is shipped with a fixed browser configuration in which plugins are disabled. Fonts are thus the most identifying feature in this context, and the best candidate to successfully identify and track anonymous Tor users.

4.2 Methodology

We follow a two-step analysis. The first step consists of an automated analysis of font detection and the second of a manual analysis of scripts and decompiled Flash source code. Font detection plays a central role in our analysis of the dataset collected by FPDetective: it is used to identify likely fingerprinting candidates in an automated fashion. We consider as likely candidates the websites that request an abnormally large number of fonts, or that have font enumeration calls in the decompiled ActionScript source code. In our analysis we classified as candidates pages that include scripts that load more than 30 fonts, or Flash files that contain font enumeration calls.

Our analysis methodology is different for the identification of Flash and JavaScript-based fingerprinters, given that font detection can be carried out by direct enumeration through Flash, and by indirect probing through JavaScript.

In the case of Flash, a dynamic analysis of font enumeration is hard due to the proprietary nature of the Flash plugin player and the use of internal font caches. At the same time, because of the byte-code nature of Flash objects, they can be straightforwardly decompiled to their original ActionScript source code. We thus decompiled the Flash objects into their corresponding ActionScript source code, selected the ones that had font enumeration calls, and manually analyzed them to understand what they do with the list of system fonts they collect. We counted a Flash object as a fingerprinter when it enumerates system fonts, collects information about the device capabilities and sends the collected fingerprint back to a remote server either by opening a socket connection or using JavaScript asynchronous calls. Furthermore we eliminated many false positives by checking Flash file URLs and domains with WHOIS lookups. For example, multimedia players that include font enumeration to be used on screen displays were eliminated by using background information.

On the other hand, due to the increased use of code obfuscation and code minimization techniques, it is, in general, very challenging to accurately perform a static analysis of JavaScript source code. We thus opted for a dynamic analysis approach, in which we intercept and record JavaScript font-probing events with the FPDetective's instrumented browsers.

As mentioned earlier, next to font-probing events, FPDetective also collects data related to other events that might be associated with fingerprinting, such as browser plugin enumeration and screen-size detection. The second part of our JavaScript-based fingerprinting analysis consisted of manually studying the identified candidates, taking into account this additional information. Specifically we checked for evidence of browser feature enumeration, including iterating over `name`, `filename`, `description` and `length` properties of browser plugins, the `enabledPlugin`, `description`, `suffixes`, and `type` properties of `mimeType` objects, and `navigator` properties such as `userAgent`, `appCodeName`, `product`, and `productSub`; `screen` object properties such as dimensions, color and pixel depth information; and properties that reveal installed toolbars such as `availTop` and `availLeft`. We classify a JavaScript file as a fingerprinter when it loads more than 30 system fonts, enumerates plugins or mimeTypes, detects screen and navigator properties, and sends the collected data back to a remote server. As we do for the Flash objects, we incorporated background information about JavaScript domain names to eliminate false positives.

We verified the correct functioning of the framework by crawling test pages that request a known set of fonts and checking them against the ones returned by FPDetective.

[12]The full comparison of the performance results can be seen here: `http://dromaeo.com/?id=197597,197598`.

5. EXPERIMENTS AND RESULTS

In the following experiments we used the FPDetective framework to crawl the top Alexa websites while searching for instances of web-based device fingerprinting. Our experiments were separated into the ones geared towards the discovery of JavaScript-based fingerprinting attempts and the remaining ones towards the discovery of Flash-based fingerprinting.

For each type of experiment, we first automatically searched for font-detection attempts and used that for selecting candidates for manual analysis. In the JavaScript experiments, we tried to find JavaScript-based font probing attempts, for which we used the number of requested fonts as a measure. In order to filter out websites that load high numbers of fonts, but do not use them for probing, we checked if websites measure the width and height of displayed text by using the number of calls to the `offsetWidth` and `offsetHeight` properties of the corresponding HTML elements.

For the Flash experiments we crawled the sites with Chromium and intercepted Flash objects with mitmproxy. We then decompiled the discovered Flash files and searched for a list of ActionScript API calls (see Appendix B for the full list) that might be relevant to fingerprinting. Subsequently, we generated a binary vector that represents the occurrence of each function call in the Flash object and used that to select objects for manual analysis.

5.1 JavaScript-Based Font probing

In this set of experiments, we crawled the top Alexa websites with FPDetective to find out the extent of the JavaScript-based font probing. In the first experiment we visited the homepages of the top Alexa 1 million websites where we waited for 10 seconds, in order to allow for the loading of remote content. In the second experiment, we visited 100,000 websites and clicked 25 homepage links from the same domain. We waited 5 seconds after each click and 10 seconds after each page load to allow for resources to load.

By analyzing the sites that are sorted by FPDetective as likely candidates of fingerprinting, we found 13 instances of JavaScript-based font-probing scripts, on a total of 404 websites. In order to ensure the accuracy of our classification, we augmented the automated, dynamic analysis with manual analysis of the source code and background information on the companies that own the domains from where scripts are served. Specifically, we checked whether the script (or another script served from the same domain) collects other high-entropy browser properties such as plugins or mimeTypes and if the company is involved in products and services that might be related to fingerprinting, e.g., device identification, analytics and anti-fraud.

Table 1 shows the results of these experiments in detail. The discovered scripts probed as many as 503 fonts and, popularity-wise, there were three websites in the top 500 Alexa sites that made use of a font-probing script.

BlueCava was discovered on the homepages of 250 Alexa websites, making it, by far, the most popular font-probing script. Moreover, it is the only one of the discovered font-probing scripts that queries different sets of fonts based on the device's operating system: 231 fonts for Microsoft Windows, 167 for Mac OS and 62 for other operating systems.

While analyzing the candidate scripts, we came across some interesting practices, such as a script that first dynamically injects itself into the page and then, after collecting the device fingerprint, removes itself from the page. This was one of the cases that motivated us to develop our tools for dynamic analysis and is discussed in more detail in Section 7.2.

Most of, but not all, the scripts that we found are served from a domain registered for an analytics and/or anti-fraud company. An intriguing case is that of Anonymizer[13], a paid anonymization service that fingerprints every visitor accessing their homepage and has a set of fingerprinting scripts that include function names such as `submitDnsInfoViaAjax`, `getClockSkew`, `getJsFontList` and `connectViaSocket`.

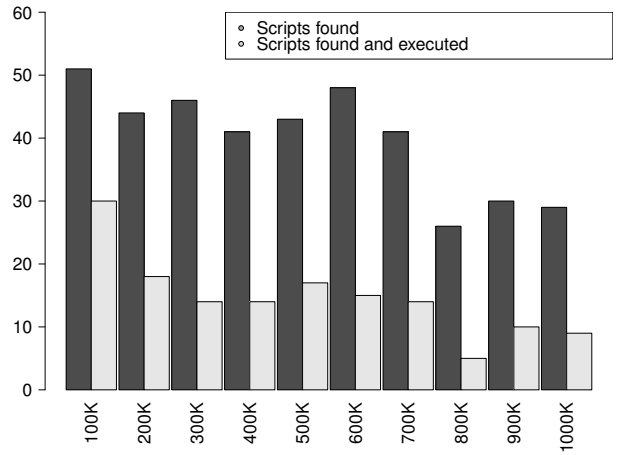

Figure 2: JavaScript-based font probing scripts on homepages of Top 1 Million Alexa sites

We also found that including third party scripts is not the only form of distribution, e.g., the font probing script from CoinBase is found as an embedded button that visitors might click to donate BitCoins to site owner.

We compared the sites that fingerprint users on the homepage, to sites that only fingerprint users on the inner pages. We did not observe any differences between these two, as evidenced by the two last columns of Table 1. This could mean that either we could not crawl the inner pages that include fingerprinting scripts not present in the homepage, or sites prefer to fingerprint users on their homepages.

The histogram in Figure 2 represents the distribution of sites using JavaScript-based font probing among the Top 1 Million Alexa sites. The histogram is divided in intervals of 100,000 sites (according to their Alexa rank) which are further divided into two bins. The darker bin refers to the total number of sites in that popularity range that served a fingerprinting script. These scripts are counted by matching previously discovered script URLs or regular expressions to URLs of HTTP requests made while visiting the site. Since we found that not all scripts probe a large number of fonts every time they are loaded, we included a second (lighter) bin to represent sites that both served a fingerprinting script

[13] `https://www.anonymizer.com/`

				Number of sites using JS-based FP		
				1M	100K	
Fingerprinting Provider	**Script name**	**Num Fonts**	**Top Rank**	**In homepage**	**In homepage**	**In inner pages**
BlueCava	`BCAC5.js`	231/167/62	1,390	250	24	24
Perferencement	`tagv22.pkmin.js`	153	49,979	51	6	6
CoinBase	`application-773a[...snipped...].js`	206	497	28	4	4
MaxMind	`device.js`	94	498	24	5	5
Inside graph	`ig.js`	355	98,786	18	1	1
SiteBlackBox	No fixed URL	389	1,687	14	10	10
Analytics-engine	`fp.js`	98	36,161	6	-	-
Myfreecams	`o-mfccore.js`	71	422	3	1	1
Mindshare Tech.	`pomegranate.js`	487	109,798	3	-	-
Cdn.net	`cc.js`	297	501,583	3	-	-
AFK Media	`fingerprint.js`	503	199,319	2	-	-
Anonymizer	`fontdetect.js`	80	118,504	1	-	-
Analyticsengine	`fingerprint.compiled.js`	93	522,447	1	-	-
				404	51	51

Table 1: Prevalence of Fingerprinting with JavaScript Based Font Probing on Top 1M Alexa sites

and probed for fonts.

5.2 Flash Based Font Enumeration

The main objective of this experiment is to investigate the extent of Flash based fingerprinting techniques on the Web. We crawled the top 10 thousand Alexa websites using FPDetective and, for each site, we visited the homepage and waited for 10 seconds to allow for resources to load completely.

We automatically decompiled all the Flash files caught by the intercepting proxy and checked the presence of functions that might be used for fingerprinting. We manually analyzed the files that include functions for known Flash-based fingerprinting techniques [18], such as querying the `Capabilities` class to collect information about the operating system and device, or opening a socket (by using `XMLSocket`) to bypass system wide proxy setting.

We also checked for the presence of function calls that might be used to send the collected information to a remote server (e.g., `sendAndLoad` or `URLLoader`) or use of `ExternalInterface.Call` or `addCallback` functions for opening JavaScript interfaces to either call JavaScript functions from the Flash file or allow page scripts to access Flash functions.

Finally, we took into account whether the collected information is sent to a remote server or used internally, by the Flash object. This helped us to filter out potential false positives, such as Flash video players that enumerate system fonts, in order to check if a font is available for use in player's on-screen display.

Table 2 shows our results for Flash-based fingerprinting. Being found on 69 sites, BB Elements, which offers tools for managing ad campaigns, is the most prevalent Flash-based fingerprinter. Piano Media is mostly found on newspaper sites and claims to employ fingerprinting for paywall enforcement. Paywalls, in this case, are used to ensure that users do not access paid content, such as newspaper or magazine articles, without subscription or payment. Usually, paywalls are combined with a limited number of free accesses, e.g., an online newspaper can offer ten free articles per user per day.

The website of the Turkish Ministry of Education (MEB) is unique in both being the only `.gov` website that employs web-based device fingerprinting and in the wealth of collected attributes from the device's hardware components. The Flash file served from this site extracts and sends detailed information about system mouse, keyboard, accelerometer, multi-touch capability, microphone, camera as well as system fonts. It should be noted that this site serves an e-School system to manage grades and other records of millions of students in Turkey. ThreatMetrix is the only discovered company which uses proxy-piercing to reveal a user's original IP address and whether she uses a proxy.

In Table 2 we can see the fingerprinters that have been discovered along with some of the functions that were called from the Flash files. We only included in the table the Flash files that have functions for font enumeration. That does not include, for instance, 50 Flash objects from one of the companies previously studied [18].

Given that we discovered several previously unknown fingerprinting scripts by extending our crawl space to 1 million for the JavaScript experiment, we plan to make a similar study for the Flash based fingerprinters.

5.3 Testing FPDetective with Fontconfig

Fontconfigis a Linux library that configures and manages the access to system fonts. In this experiment we used debugging functionality of fontconfig to test our framework's accuracy.

We crawled the Alexa Top 10,000 sites and compared the set of font requests made by each web site to ensure that FPDetective is not missing or overcounting font events. The comparison of the two measurements indicated that we neither miss nor overcount the font load events.

5.4 Prevalence of Fingerprinting

With FPDetective we found 404 sites in the Alexa top million pages that fingerprint visitors on their homepages using JavaScript-based font probing. These scripts are served by 13 different fingerprinting providers, of which only a few had been identified in prior research.

Although 404 out of 1,000,000 can be thought of as a very low frequency, we would like to note that the results presented here should be taken as lower bounds, as our crawlers cannot reach pages that are placed after forms including CAPTCHAs or similar obstacles. Moreover, Flash-based fingerprinting was present the homepages of 97 out of the

Fingerprinting Provider	Num of Sites	Top Rank	Flash Cookies	Proxy Detection	HW and OS Profiling	Interaction with JS
BB Elements	69	903	✓			✓
Piano Media	12	3,532	✓		✓	✓
Bluecava	6	1,390	✓			✓
ThreatMetrix	6	2,793		✓	✓	
Alipay	1	83	✓		✓	✓
meb.gov.tr (Turkish Ministry of Education)	1	2,206	✓		✓	✓

Table 2: Flash Fingerprinting objects found on Top 10K Alexa websites

top 10,000 sites, indicating that Flash-based fingerprinting is more prevalent. This is possibly because of its extended capabilities for font enumeration, proxy detection and its widespread browser support.

6. EVALUATION OF FINGERPRINTING COUNTERMEASURES

In this section we briefly analyze two tools that can be employed to counter fingerprinting: the Tor Browser and the FireGloves Firefox extension. We also evaluate whether the Do Not Track header is being respected by fingerprinters.

6.1 Tor Browser

The Tor Browser is part of the software bundle that is used to access the Tor anonymity network [5], a popular service that is currently used daily by more than 800,000 people to anonymously browse the web. Tor relays communications over three routers located in different parts of the world. The communications are encrypted in layers (onion routing [8]) to prevent any single Tor router from linking the source and destination of a data stream.

The Tor Browser incorporates strong defenses to counter the fingerprinting techniques described by the Panopticlick study, as its design goals include enabling user anonymity and preventing the linkability of browsing sessions. Given that the Tor Browser has a limited user base compared to web users in general, even a partial fingerprint might be enough to uniquely distinguish a Tor user. Thus, there is a need to eliminate fingerprintable differences among Tor Browsers, so that Tor users remain indistinguishable from each other. In fact, *cross-origin fingerprinting unlinkability* is listed as a privacy requirement for the Tor Browser[14]. For this reason the Tor Browser is shipped with fixed settings that provide almost no browser properties that could be exploited to extract distinguishing features.

As described in Section 4, fonts are operating-system-dependent, and thus a good candidate attribute for distinguishing and recognizing users. To limit font-based fingerprinting, the Tor Browser caps the number of fonts that a page can request and load. However, for usability reasons, `@font-face` rules are exempted from these limits. Upon inspection of the Tor Browser source code, we discovered that the local fonts loaded by `@font-face` CSS rules are also exempted from the Tor Browser's font-per-document cap, and that it is possible to load an unlimited number of system fonts using the `local()` value of the `@font-face` rule's `src` descriptor.

Furthermore, if a font is not installed locally, it may be requested from the `src/url` property of the font-face rule. This is effectively communicating the lack of this font to the server without using JavaScript. Note also that one local font-face rule can be used to report the `status(not-found)` of more than one font by chaining font-face `local()` properties, for example:

```
@font-face { font-family: Font1;
             src: local("Font2"), local("Font3"),
                  url("Font1-2-3-NotFound.ttf");
           }
```

We immediately communicated the vulnerability to the Tor Bug Tracker and the issue is fixed in the upcoming (2.4) version with a patch that disables the use of the `local()` property[15].

6.2 Firegloves

Firegloves [3] is a proof-of-concept browser extension for Mozilla Firefox that was created for research purposes. In order to confuse fingerprinting scripts, Firegloves returns randomized values when queried for certain attributes, like the screen resolution, the platform on which the browser is running and the browser's vendor and version. Additionally, Firegloves limits the number of fonts that a single browser tab can load and reports false dimension values for the `offsetWidth` and `offsetHeight` properties of HTML elements to evade JavaScript-based font detection.

We evaluated the effectiveness of Firegloves' as a countermeasure to fingerprinting, and discovered several shortcomings. For instance, instead of relying on `offsetWidth` and `offsetHeight` values, we could easily use the width and the height of the rectangle object returned by `getBoundingClientRect` method, which returns the text's dimensions, even more precisely than the original methods[16]. This enabled us to detect the same list of fonts as we would without the Firegloves extension installed. Surprisingly, our probe for fonts was not limited by the claimed cap on the number of fonts per tab. This might be due to a bug, or to changes in the Firefox extension system that have been introduced after FireGloves, which is not currently being maintained, was first developed.

Although Firegloves spoofs the browser's user-agent and platform to pretend to be a Mozilla Firefox version 6 running on a Windows operating system, the `navigator.oscpu` is left unmodified, revealing the true platform. Moreover, Firegloves did not remove any of the new methods introduced in later versions of Mozilla Firefox and available in

[14] `https://www.torproject.org/projects/torbrowser/design/#fingerprinting-linkability`

[15] `https://trac.torproject.org/projects/tor/ticket/8455#comment:3`

[16] `https://developer.mozilla.org/en-US/docs/DOM/element.offsetWidth`

the `navigator` object, such as `navigator.mozCameras` and `navigator.doNotTrack`. Finally, since Firegloves cannot change the APIs available to Flash, a Flash application can still discover the real operating system and the real screen resolution.

Overall, while Firegloves is trying to protect users from fingerprinting, its detectable presence on the users' browsing environments may actually make them more uniquely identifiable: since Firegloves is installed by only 1,750 users, its successful detection makes the user much more unique than if it was not present at all.

Our findings are in line with prior results showing that user-agent-spoofing extensions can be straightforwardly discovered and bypassed [18]. These findings illustrate the difficulty of effectively protecting users against fingerprinting, and indicate that counter-measures short of perfect may result in a net loss of privacy for their adopters – as their devices become more easily fingerprintable through these.

6.3 Do Not Track

The Do Not Track (DNT) HTTP header field allows users to signal their tracking preferences to websites. DNT is currently being standardized by the W3C under the name "Tracking Preference Expression" and it has already been adopted by most modern browsers[17].

We set the DNT header to 1 in the PhantomJS browser and visited the websites identified as performing fingerprinting in our previous experiments. For all of these pages, we obtained the same results with respect to the number of fonts probed and other browser properties accessed, suggesting that DNT preferences are ignored by fingerprinters.

7. DISCUSSION

7.1 Uses of Fingerprinting

Using FPDetective, we were able to identify companies that are engaging in what we call web based device fingerprinting. While it is not possible to infer the purposes for which fingerprinting is being put to use by these companies, we take a moment to reflect on their fingerprinting related practices.

The majority of the companies serving the fingerprinting scripts explicitly state on their websites, in press releases, and in various social media channels that they are successfully deploying (device) fingerprinting. In contrast, the websites that employ the services of these companies rarely disclose in their privacy policies that they are fingerprinting their users' devices, let alone mention that they are gathering the information that FPDetective revealed them to be collecting. In some cases, we detected fingerprinting scripts that were embedded in ad banners. It is unclear whether first parties serving these ad banners are aware of the existence of these embedded fingerprinting scripts.

Companies express that they deploy device fingerprinting in the context of a variety of web services. The spectrum of use cases include fraud detection, protection against account hijacking, anti-bot and anti-scraping services, enterprise security management, protection against DDOS attacks, real-time targeted marketing, campaign measurement, reaching customers across devices, and limiting number of access to services. While most companies will specialize either in 'fraud detection and web security' or 'marketing applications', MaxMind is an example of a company that uses their data collection to provide services for both uses. MaxMind also stands out with their very explicit documentation of their personal and non-personal data collection, their processing activities and their commitment to make their practices transparent.

Further material from companies describing the use cases suggest that the data models in the databases also vary. Fraud detection companies often speak of "device reputation databases" with profiles for "billions" of devices that are rich enough to provide "intelligence" to companies about the security risks of these devices. The variety in mentioned uses ranging from "adjusting security policies based on the device a person is using" to "identifying and blocking botnets that easily switch IP addresses" suggests that these are rich device-centered databases. On the other hand, marketing companies express that they are able to identify user-behavior across websites and devices, suggesting that they feed fingerprinting data into "customer"-centered database systems that are heavy on analytics, and can link different devices to a single user across websites.

In-house applications seem to be mainly concerned with whether the same user is using multiple devices to access the same service, or limiting the number of times a specific service is accessed, e.g., when filling out surveys. Such applications seem to limit data collection using fingerprinting to the given website and are less concerned with analytics and intelligence applications.

A perplexing case among in-house applications is that of Anonymizer Inc., a company that presents itself as "the global leader in online privacy, anonymity, and identity protection solutions for over 17 years."[18] The script is served from the domain privacytool.org (owned by Anonymizer Inc.), a site where users can test whether they are anonymous online. The privacytool.org site clearly explains that to perform the test a Java applet will run on the user's computer, describes the information that will be gathered, and explicitly states that *"Data obtained from the browser like lists of plug-ins or fonts can be used to identify your computer."*[19] Users must click a link placed below this information in order to run the applet.

We found, however, that fingerprinting scripts from privacytool.org are also present in the homepage of anonymizer.com, another site owned by Anonymizer Inc. that, paradoxically, offers anonymity as a service. The anonymizer.com privacy policy states: *"When you navigate our Web site, Anonymizer will gather certain information such as your Internet Protocol address, browser type, browser language, and the date and time of your visit. We may place a cookie on your computer [...]"* The policy includes further information about cookies but, in contrast to the privacytool.org notice, it does not mention at any point the execution of fingerprinting scripts, or that it collects information such as the list of installed fonts, DNS server information, or the real IP address if the user is connected through a proxy. Finally, note that while privacytool.org offers informed choice to its users, who may voluntarily execute the script, the fingerprinting scripts that run in the

[17] `http://www.w3.org/TR/2013/WD-tracking-dnt-20130430/`

[18] `https://www.anonymizer.com/company/`

[19] `http://privacytool.org/AnonymityChecker/index.jsp`

anonymizer.com homepage are invisible to users and run by default.

7.2 Visibility of Fingerprinting

The majority of web users have difficulties in grasping what cookies are, whether they are enabled, their threats to their privacy and how to manage them [14]. This situation is worse in the case of third-party and covert cookies, which have been found to essentially be invisible to end-users. Arguably, user perception of web based device fingerprinting is comparably, if not more, invisible to the users.

In all the cases we encountered, there were no visible effects of fingerprinting, and the users were not informed that they were being fingerprinted. Thus, the only way for users to discover that their devices are being fingerprinted is to manually examine the source of the page and all the embedded JavaScript and Flash objects.

In one distinct case, while verifying the results of FPDetective, we discovered that a company which sells fingerprinting products for anti-bot and anti-scraping services, was deleting the fingerprinting script from the page's DOM after the script had executed and collected the fingerprint. Thus, the only way to identify this fingerprinter is to breakpoint through the execution of JavaScript code and witness the loading and unloading of the fingerprinting code. This requires an in-depth understanding of how JavaScript is executed and the principles of debugging programs, which the vast majority of users are not likely to possess. Moreover, this anti-debugging technique is reminiscent of the techniques employed by JavaScript malware when trying to evade detection by analysts and high-interaction honeypots [10].

7.3 Is fingerprinting a matter of privacy?

Our findings suggest that this issue may require further technical and legal attention with respect to privacy. Yet, most of the companies whose fingerprinting activities were detected by FPDetective expressly distance their practices from any consequences they might have for people's privacy.

A number of companies argue that they do *not* collect PII (Personally Identifiable Information) in the process of distinguishing "the good, the bad and the ugly" on the web. While fingerprinting may not require PII, the use cases described by fingerprinting companies on their webpages suggest that they use device information to track, profile, and shape the future web experience of the tracked users (as well as bots). For example, MaxMind offers online retailers a service to check on their customers' fraud scores based on 31 "non-PII" attributes, including IP address, shipping address, non-salted hashes of email addresses and passwords, and credit card BIN number. The computation of the customers' score is based on things like the 'riskiness of the country of origin', 'proxy use', 'free webmail use', and 'bank checks'. These are matters closely related to privacy concerns expressed by users about uninformed monitoring of web usage, constraints on informational self-determination, and discrimination [21].

Further, by focusing on "device" identification and, especially in fraud detection cases, claiming that they are concerned only about "bots", companies express that tracking "persons" is not the object of their interest. In this worldview, fingerprinting is nothing more than (security) scripts collecting data based on socially invisible interactions that are irrelevant to individual privacy. Further, especially in the case of fraud detection, companies often argue that fingerprinting is implemented for the protection of the end-users' quality of service. These two framings, i.e., "fingerprinting is all about devices" and "we track these devices for user convenience", make it very difficult to demand a response to the privacy issues that may be raised with respect to device fingerprinting and the use of the databases populated using fingerprinting. As such fingerprinting practices proliferate, device IDs may come to "represent" users in databases, instead of PII. Hence, classical conceptions of PII may not be sufficient to grasp the social and ethical concerns associated fingerprinting and related databases. We hope that this paper, by virtue of making web based fingerprinting more visible, will contribute to better understanding what privacy issues may be at stake and to challenging the framing of web based device fingerprints as non-PII.

In the context of the US, it may be worth discussing whether a static list of attributes that count as PII is sufficient to draw a reasonable boundary on which "personal data" should be subject to protection. Device fingerprinting underlines that data is identifiable based on context; in other words, identifiability may result from processing seemingly non-identifiable information [22]. If we accept this argument, then attention needs to be paid to risks associated with: linking of the reputation and device fingerprint databases to individuals; undesirable and unacceptable uses of these datasets for determining "the good, the bad, and the ugly" of the web; the security of these datasets; and the opacity of fingerprinting practices to the general public, as well as to individual device owners.

It is also currently unclear whether there is a responsibility to inform the owners that their devices are being fingerprinted, and if so, who has the responsibility to inform the users. For example, the privacy policy of `articlesbase.com` explicitly indicates all the "non-personal information" that the site collects. However, in this rather detailed and readable list, the site does not state that they are fingerprinting the user's device or that they are probing fonts. One could argue, since the font probing scripts are not served by `articlesbase.com` but by `siteblackbox.com`, that it is the responsibility of the latter to inform the users. However, in their documentation of the articlesbase case [20], no references are made to fingerprinting of users and we were unable to locate a privacy policy on the website. Similar issues are likely to arise when it comes to honoring DNT preferences.

8. CONCLUSION

User tracking is becoming pervasive as advertisers and tracking companies seek to refine their targeting, detect fraud, or offer new services. While most of today's tracking is done through third-party cookies, prior research has shown that browser and system attributes can be used to uniquely identify devices through fingerprints. Even though these fingerprints are less accurate than stateful identifiers such as cookies, their main advantage is that device fingerprinting is harder to detect and to defend against.

In this paper we presented FPDetective, a fingerprinting-detection framework that identifies web based fingerprinters. Using FPDetective, we performed a large-scale crawl of the Internet's most popular websites, and showed that the adoption of fingerprinting is significantly higher than what previous research estimated. Among others, we identi-

fied large commercial companies involved in fingerprinting, a complete disregard towards the DNT header, and the use of anti-debugging techniques, most commonly associated with JavaScript malware. Moreover, we showed that dedicated fingerprinters can bypass existing privacy-protecting technologies.

Overall, our findings demonstrate that web fingerprinting is a real and growing issue, deserving the attention of both policymakers and the research community. We hope that our framework, which is freely available to other researchers and can easily be extended to conduct further studies, will contribute to addressing this issue by providing a means to shed light on web fingerprinting practices and techniques.

9. ACKNOWLEDGEMENTS

The authors would like thank the anonymous reviewers, Carmela Troncoso, Ashkan Soltani, Nessim Kisserli and Tom van Cutsem for their valuable comments; Vicenç Torra for enabling the collaboration; and Danny De Cock for his hardware support. For KU Leuven, this research was supported by the projects IWT SBO SPION, iMinds CoMobile, FWO G.0360.11N, FWO G.0686.11N, GOA TENSE (GOA/11/007), and EU FP7 NESSoS, WebSand and Strews; the B-CCENTRE; and the Research Fund KU Leuven. For IIIA-CSIC, this research was supported by the projects EU FP7 grant agreement number 262608, the Spanish MEC projects ARES (CONSOLIDER INGENIO 2010 CSD2007-00004), eAEGIS (TSI2007-65406-C03-02), and COPRIVACY (TIN2011-27076-C03-03). For NYU, this research was supported by the Intel Science and Technology Center - Social Computing. Marc Juarez was partially funded by the LLP Erasmus Programme of the Commission of the European Communities during his stay at KU Leuven.

10. REFERENCES

[1] Nmap - Free Security Scanner For Network Exploration & Security Audits. `http://www.nmap.org`.

[2] A. Barth. Blink: A rendering engine for the Chromium project. `http://blog.chromium.org/2013/04/blink-rendering-engine-for-chromium.html`.

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

[4] A. Cortesi. mitmproxy: a man-in-the-middle proxy. `http://mitmproxy.org/`.

[5] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320. USENIX, 2004.

[6] J. R. Douceur. The sybil attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002.

[7] P. Eckersley. How Unique Is Your Browser? In *Proceedings of the 10th Privacy Enhancing Technologies Symposium (PETS)*, pages 1–17, 2010.

[8] D. Goldschlag, M. Reed, and P. Syverson. Hiding routing information. In *Information Hiding*, pages 137–150, 1996.

[9] D. Jang, R. Jhala, S. Lerner, and H. Shacham. An empirical study of privacy-violating information flows in JavaScript Web applications. In *Proceedings of CCS 2010*, pages 270–283, Oct. 2010.

[10] A. Kapravelos, M. Cova, C. Kruegel, and G. Vigna. Escape from monkey island: Evading high-interaction honeyclients. In *Proceedings of the 8th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 124–143, 2011.

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

[12] J. R. Mayer. Any person... a pamphleteer. Senior Thesis, Stanford University, 2009.

[13] J. R. Mayer and J. C. Mitchell. Third-party web tracking: Policy and technology. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 413–427, 2012.

[14] A. D. Miyazaki. Online privacy and the disclosure of cookie use: Effects on consumer trust and anticipated patronage. *Journal of Public Policy & Marketing*, 27(1):19–33, 2008.

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

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

[17] M. Mulazzani, P. Reschl, M. Huber, M. Leithner, S. Schrittwieser, and E. Weippl. Fast and reliable browser identification with javascript engine fingerprinting. In *Web 2.0 Workshop on Security and Privacy (W2SP)*, May 2013.

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

[19] Ł. Olejnik, C. Castelluccia, and A. Janc. Why Johnny Can't Browse in Peace: On the Uniqueness of Web Browsing History Patterns. In *the 5th workshop on Hot Topics in Privacy Enhancing Technologies (HOTPETS 2012)*.

[20] SiteBlackBox. Case studies: Articlesbase.

[21] B. Ur, P. G. Leon, L. F. Cranor, R. Shay, and Y. Wang. Smart, useful, scary, creepy: perceptions of online behavioral advertising. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 4:1–4:15, New York, NY, USA, 2012. ACM.

[22] I. Van Der Ploeg. *Keys To Privacy. Translations of "the privacy problem" in Information Technologies*, pages 15–36. Maastricht: Shaker, 2005.

[23] T.-F. Yen, Y. Xie, F. Yu, R. P. Yu, and M. Abadi. Host fingerprinting and tracking on the web:privacy and security implications. In *Proceddings of the 19th Annual Network and Distributed System Security Symposium (NDSS)*, 2012.

[24] N. C. Zakas. How many users have JavaScript disabled? `http://developer.yahoo.com/blogs/ydn/many-users-javascript-disabled-14121.html`.

[25] M. Zalewski. p0f v3 (version 3.06b). `http://lcamtuf.coredump.cx/p0f3/`.

FingerPrinter	Script URL
BlueCava	http://ds.bluecava.com/v50/AC/BCAC5.js
Perferencement	http://tags.master-perf-tools.com/V20test/tagv22.pkmin.js
CoinBase	https://coinbase.com/assets/application-773afba0b6ee06b45ba4363a99637610.js
MaxMind	http://device.maxmind.com/js/device.js
Inside graph	http://inside-graph.com/ig.js
SiteBlackBox	(No fixed URL)
Analytics-engine	http://sl4.analytics-engine.net/detector/fp.js
Myfreecams	http://www.myfreecams.com/mfc2/lib/o-mfccore.js
Mindshare Tech.	pomegranate.js(Nofixeddomain)
Cdn.net	http://cdn-net.com/cc.js
AFK Media	http://gmyze.com/0.4.1.1/js/fingerprint.js
Anonymizer	https://www.privacytool.org/AnonymityChecker/js/fontdetect.js
Analyticsengine	http://dpp750yjcl65g.cloudfront.net/analyticsengine/util/fingerprint.compiled.js
BBelements	http://go.eu.bbelements.com/flash/bbnaut.swf
Piano Media	http://mp.pianomedia.eu/bucket/novosense.swf
Bluecava	http://lookup.bluecava.com/flash/guids[2|3].swf
ThreatMetrix	https://h.online-metrix.net/fp/fp.swf?org_id=...&session_id=... (also served from other domains)
Alipay	http://img.alipay.com/common/um/lsa.swf
MEB	http://meb.gov.tr/KZneA1akxW/502758.swf

Table 3: URLs of Fingerprinting JavaScript and Flash Files

APPENDIX

A. LIST OF FINGERPRINTING URLS

Table 3 shows the location of each fingerprinting script discovered, separated by the respective fingerprinting companies.

B. ACTIONSCRIPT CALLS

The following list enumerates the ActionScript calls that FPDetective searches for in the decompiled Flash files.

- `enumerateFonts` (only with argument true)
- `getFontList`
- all `Capabilities` class properties and methods, including
  - `version`
  - `manufacturer`
  - `serverString`
  - `language`
- `screenDPI`
- `screenResolutionX`
- `screenResolutionY`
- `getTimezoneOffset`
- `getLocal`
- `XMLSocket`
- `Math.min, Math.max`
- `ExternalInterface.call`
- `ExternalInterface.addCallback`
- `sendAndLoad`
- `URLLoader`
- `navigateToURL`
- `loadMovie`
- `createUID`
- `getUrl`
- `allowDomain`
- `allowInsecureDomain`
- `loadPolicyFile`
- `URLRequest`
- `LoadVars`
- `md5, sha256, sha384, sha512`