FP-Block: usable web privacy by controlling browser fingerprinting

Christof Ferreira Torres[1], Hugo Jonker[2] and Sjouke Mauw[1]

[1] CSC/SnT, University of Luxembourg
[2] Open University of the Netherlands

Abstract. Online tracking of users is used for benign goals, such as detecting fraudulent logins, but also to invade user privacy. We posit that for non-oppressed users, tracking within one website does not have a substantial negative impact on privacy, while it enables legitimate benefits. In contrast, *cross-domain* tracking negatively impacts user privacy, while being of little benefit to the user.
Existing methods to counter fingerprint-based tracking treat cross-domain tracking and regular tracking the same. This often results in hampering or disabling desired functionality, such as embedded videos. By distinguishing between regular and cross-domain tracking, more desired functionality can be preserved. We have developed a prototype tool, *FP-Block*, that counters cross-domain fingerprint-based tracking while still allowing regular tracking. *FP-Block* ensures that any embedded party will see a different, unrelatable fingerprint for each site on which it is embedded. Thus, the user's fingerprint can no longer be tracked across the web, while desired functionality is better preserved compared to existing methods.

1 Introduction

Online activities play an ever-growing role in everyday life. Consequently, companies are increasingly tracking users online [14]. There may be various reasons for such tracking, such as fraud prevention by identifying illegitimate usage attempts [16], suggesting related content, and better targeting advertisements. Where such tracking remains confined to the tracker's own website, the balance between privacy and functionality is (arguably) satisfied: the website learns a user's browsing habits *on that particular website*, which helps to improve the website *for this user*. We will call this type of tracking *regular tracking*.

However, some companies offer online services that are embedded on a large number of websites. Examples of such services are social sharing buttons, popular JavaScript libraries, and popular web analytics services. Thanks to this ubiquitous embedding, such companies can track users over large portions of the web. According to various studies, plenty of different companies are embedded on a sizable[3] portion of the Web. For example, consider the Facebook "Like" button.

[3] E.g. penetration rates for top 1 million sites according to BuiltWith.com (October 2014): *DoubleClick.net* 18.5%, *Facebook Like button* 15.6%, *Google Analytics* 46.6%

The embedding site includes a piece of code that triggers the user's browser to contact the Facebook servers to download the button. As browsers are made to explain where a request originated (the HTTP `Referer` field), the browser will tell Facebook exactly which URL triggered this request each time. This enables Facebook to track the user across the web [15], irrespective of whether or not the user even has a Facebook account. We will call this type of tracking *third-party tracking.*

This tracking can be done using an HTTP cookie, but even if such (third party) cookies are blocked, it is possible to infer a set of attributes (screen resolution, HTTP user agent, time zone, etc.) that are often sufficient to uniquely identify the user [5]. Note that such attributes were intended to benefit the user, e.g. to present her the mobile version of a site when browsing from a phone, or to present the page in the user's preferred language. Yet even though personalia such as name or age are not explicitly revealed, the tracker can learn far more about the users than one realises[4].

Identifying users by such means is called "fingerprinting". Much like a fingerprint belongs to one unique individual, a fingerprint of communication with a server belongs to a unique browser.

Existing countermeasures combat such fingerprint-based tracking with little regard for the impact on the the user experience. The goal is then to find an approach that ensures a better balance between user experience (that is: less impact on desired embedded contents) and tracking. We address this with the concept of *web identity*: the set of fingerprintable browser characteristics. A web identity is generated for the main site visited (e.g., `bbc.com`), and that identity is then also used for all interaction with embedded contents on that site (e.g. videos, social media buttons, etc.). If the user visits a different site (e.g., `cnn.com`), a different identity is used (the web identity for `cnn.com`). Thus, a party embedded on both sites will first see the web identity for `bbc.com`, and later the web identity for `cnn.com`. As we ensure that the generated identities are distinct, the two visits can no longer be linked by means of their fingerprint. We focus on fingerprinters that aim to re-identify as many users as possible. We do not target tools that are seeking to track any one specific individual.

Contributions. The main contributions of our research are the following:

1. We argue that a distinction be made between regular and cross-domain tracking; we observe that current anti-tracking tools do not make this distinction.
2. We introduce the notion of a web identity to generalize the rather dynamic notion of a fingerprint. We propose *separation of web identities* as an approach to fingerprint privacy that prevents cross-domain tracking, while allowing regular tracking.
3. We have developed a prototype browser extension, *FP-Block*, that supports the automatic generation and management of web identities. A user's web identity remains constant across all his visits to the same domain, while his

[4] E.g. by data aggregation, a supermarket can infer if a customer is pregnant, and estimate her due date (Forbes.com, 2012).

web identities for different domains are not related. The tool has a user interface that shows which third parties are embedded in the current page, which fingerprintable attributes are requested by the website and how these are spoofed by the tool. This allows the user to fine-tune the tool for individual sites.
4. By deobfuscating the code of the most relevant fingerprinters, we compiled an up-to-date list of the attributes used by current fingerprinters. Further, based on a literature study, we compiled a list of the attributes affected by the most relevant anti-tracking tools. These lists guided our design and helped us to compare our work to existing tools.
5. We have tested our tool against six fingerprinters in two different scenarios: first-party fingerprinting and third-party fingerprinting. In all cases, our tool was successful in affecting the computed fingerprint.
   As final validation, we tested our tool against a commercial fingerprinter's online tracking ID. Again the tool was successful: the tracking ID without the tool was different from the ID when running the tool.

2 Related work

We distinguish related work between work on fingerprinting and work on countermeasures against tracking and fingerprinting.

2.1 Fingerprinting

The possibility of remotely inferring characteristics of devices has been known for some time. E.g. Kohno et al. [6] use TCP clock skew to remotely fingerprint devices. Eckersley [5] was the first to draw significant attention to the problem of fingerprinting web browsers. His fingerprinting algorithm returned a unique fingerprint for 83.6% of the browsers considered. Subsequent papers established how to fingerprint other aspects such as the user-agent string and IP address [17], the HTML5 "canvas" element [10], the used Javascript engine [9,11]), the fonts present [3], etc. Other work has suggested approaches to combine several fingerprintable attributes to arrive at unique fingerprints (e.g. [3,17]). Using such techniques only on one site does not constitute a large invasion of privacy. However, as Mayer and Mitchel [8] and Roosendaal [15] have pointed out, social plugins are embedded on many pages, which allows the social network to track users across the Internet. Thanks to fingerprinting, such tracking does not even require an account – anyone with a unique fingerprint can be traced.

Several recent studies have investigated the practice of fingerprinting in more detail. Acar et al. [2] introduced a framework to detect online fingerprinters by detecting activities typical of fingerprinters (e.g. requesting values of certain attributes, enumerating fonts, etc.). Their detection algorithm found several fingerprinting techniques in action. Many of these were by companies offering fingerprinting services to others: the fingerprinters were either embedded by the visited site, or by a third party such as inside a third-party advertisement. In a followup study, Acar et al. [1] investigate the spread of three tracking mechanisms, amongst which HTML5 canvas fingerprinting. They propose three heuristics to estimate whether a canvas is being used to fingerprint; we adopt these criteria in our tool. This paper also shows that canvas fingerprinting is being used by a popular 3rd party service (AddThis.com). Nikiforakis et al [13] provide a detailed study of the fingerprinting techniques of three online fingerprint services (BlueCava, Iovation and ThreatMetrix). They found that fingerprinting can be tailored to the detected browser, e.g. when detecting a browser as Internet Explorer, the fingerprint would include Internet Explorer-specific attributes. Moreover, Flash was used by all three to enrich the established fingerprint. Flash allows access to many similar attributes as JavaScript, but may give a more detailed response (e.g. including major and minor kernel version number). Moreover, Flash ignores a browser's proxy settings for client-server communication. This allows the fingerprint services to detect if the user is behind a proxy or not, and correlate the IP address of the proxy with that of the user.

2.2 Countermeasures

Several existing plugins directly aim to stop trackers, including commercially developed ones (e.g. Ghostery, AVG Do Not Track, etc.), and academically developed ones, such as FireGloves [3], ShareMeNot [14], PriVaricator [12], etc. Some of these work by blocking trackers, either by using a predefined blacklist or by updating the blacklist on the fly using heuristic rules. Krishnamurthy and Wills [7] argue that blacklists are prone to being incomplete (not all trackers blocked), and therefore ultimately fail at blocking trackers. A similar argument holds against heuristically updated blacklists: the next tracker may well use tracking methods not covered by the heuristics, and so escape notice. More damning, however, Mowery et al. [9] show how an attacker can detect a blacklist and use its content as an additional fingerprintable attribute.

Other plugins work by faking attributes (FireGloves, PriVaricator). As [5,13] point out, such spoofing may lead to inconsistencies that paradoxically make the user stand out more. Indeed, Acar et al. [2] argue this is the case for FireGloves-equipped browsers. Even more strongly, Nikiforakis et al. [13] advice against the use of any user-agent-spoofing extension.

We add the following observations. First of all, fingerprinting can be done both passively and actively. Passive fingerprinting uses attributes inherent in the communication (e.g. the order of HTTP headers), while active fingerprinting executes attribute-gathering scripts on the client-side (e.g. determine JavaScript engine speed). Anti-tracking tools should take into account both mechanisms. Secondly, anti-tracking tools should consider "fingerprint consistency", i.e., the extent to which a fingerprint is perceived as genuine. By carefully tailoring the spoofing system to account for known fingerprinters, their consistency checks can be satisfied. Secondly, fingerprinting trackers take a fingerprint of each user, and aim to link a new fingerprint to an existing fingerprint. If the new fingerprint is sufficiently different, this link can*not* be made – irrespective of how unique this new fingerprint is. PriVaricator [12] uses similar ideas to these. In contrast, our approach addresses both active and passive fingerprinting, uses real-world browser statistics to generate consistently spoofed fingerprints, and works as a browser extension instead of modifying the browser itself.

The Tor Browser[5], widely recognised as providing the best privacy, uses another approach: it aims to keep one single set of attribute values across all its instances. In ideal circumstances, this means that no two Tor Browser instances can be distinguished from each other, however, users can install plugins or tweak settings which undo this protection.

The Tor browser is strongly focused on preserving privacy, providing a level of privacy believed to be sufficient for use under intense state scrutiny. This strong privacy comes at the expense of some usability, most notably, that all data is sent via an onion routing network. We posit that the strong privacy offered by the Tor Browser exceeds the needs of non-oppressed users. Therefore, we will focus on the prevention of third-party tracking.

3 Determining the Fingerprint Surface

In this section, we determine the relevant set of characteristics that are used to fingerprint a user. Determining the full set of characteristics, the *fingerprint surface*, is not practically possible. For example, as a plugin can change the browser's behaviour in ways that can be detected, constructing a complete set of characteristics necessarily requires examining all browser plugins. Therefore, we focus pragmatically on that part of the fingerprint surface that is being used by fingerprinters. This implies that we use their definition of identity. As fingerprinters equate a fingerprint of the device with a user identity, we will use this (somewhat imprecise) abstraction as well.

To establish the fingerprint surface, we determine for four major commercial fingerprinters which characteristics they use to fingerprint, i.e., their *fingerprint vector*. We also examine four major anti-fingerprint tools and determine which part of the fingerprint surface they consider. Together, this gives us a practical approximation of the fingerprint surface. *FP-Block* then is built to ensure no two fingerprints are fully coincide within this approximation.

3.1 Limitations of Preventing Fingerprint Tracking

There are two approaches to prevent fingerprint-based tracking: blocking trackers, and spoofing attribute values. However, neither by itself suffices to prevent tracking, while both impact user experience.

Blocking fingerprinters. A naive solution is to block any third party content. However, many webpages embed such content, without which the webpage renders incorrectly (e.g. content delivery networks, embedded video). Other third party content helps to sustain and improve the site (counters, advertisers, analytics,...). The impact of blocking all third party content on usability will be

[5] `https://www.torproject.org/projects/torbrowser.html.en`

far too great. Blocking known major trackers would provide some protection. However, such a list will remain incomplete [7], irrespective of updates. Thus, blocking by itself cannot fully address third party tracking without impacting on desired functionality.

Spoofing attribute values. The attribute values that make up the fingerprint may be faked. This changes the fingerprint, which in turn changes what the tracker infers as the identity. There are many browser plugins that randomise a (sub)set of attributes. However, not all attributes can be faked without substantially impacting user experience (e.g. JavaScript engine speed). Moreover, faked data often makes the user's fingerprint stand out *more* [5,13], making the fingerprint more easy to recognise. Finally, new fingerprintable characteristics keep on being discovered. Thus, it is impossible in practice to determine and spoof the full set of fingerprintable characteristics.

Thus, determining the entire fingerprint surface is impossible. As such, no anti-fingerprint tool can prevent all fingerprinting. This motivates a pragmatic approach: determine which characteristics are used by a set of fingerprinters, and focus on those.

3.2 Fingerprint Vectors

To determine which characteristics are commonly used to fingerprint users, we determine the fingerprint vectors of several widely-used fingerprinters. We analyze three commercial trackers: BlueCava (BC), IOvation (IO), and ThreatMetrix (TM). Note that this part of our analysis builds on and expands the results of [13]. We furthermore analyze one social plugin that fingerprints users: AddThis (Add). We complement this with characteristics gathered from Eckersley's seminal work [5] (Pan) and from an open source fingerprint library, FingerPrintJS (FPjs). The results are presented in the left part of Table 1 (legend and footnotes in Table 2).

In Table 1, each row denotes a fingerprintable characteristic. A $\surd$ sign indicates that the fingerprinter uses this characteristic in determining a fingerprint. Characteristics marked with ✓ were previously reported in [13]; those marked with — were reported, but are no longer used.

Recall that in this work we only target the HTTP and JavaScript layer of the communication stack. Fingerprinting can also occur at other layers, e.g. at the TCP/IP layer, or at the Flash layer.

For *FP-Block* we only address HTTP and JavaScript characteristics that are used by one ore more of the six analyzed fingerprinters.

3.3 Fingerprint Surface

To determine the minimum set of characteristics that needs to be protected, we investigate four popular anti-fingerprinting tools: FireGloves [3] (FG), the Tor

	Fingerprinters						**Countermeasures**					
Attribute	Pan	BC	IO	TM	Add	FPjs	FG	Tor	PV	RAS	**FPB**	
Plugin Enumeration	✓	✓	✔	✓	✔	✔	✔	✔	✔		✔	
Font Detection	✓	✓		✓			✔	✔	✔	✔	✔	
User-Agent	✓	✓	✓	✓	✔	✔	✔	✔		✔	✔	1
HTTP Header Accept	✓											1,5
HTTP Header Accept-Charset	✓							✔				1,5
HTTP Header Accept-Encoding	✓									✔	✔	1
HTTP Header Accept-Language	✓							✔		✔	✔	1
Screen Resolution	✓	✓	✓	✓	✔	✔	✔	✔		✔	✔	
Timezone	✓	✓	✓	✓	✔	✔	✔	✔		✔	✔	
Browser Language		✓	✓	—	✔	✔	✔				✔	
OS & Kernel Version		✔	✔	✓	✔	✔	✔	✔		✔	✔	
DOM Storage	✓	✔	✔	✔	✔	✔				✔	✔	
IE userData	✓	✔										2
Java Enabled	✔				✔						✔	
DNT User Choice		—			✔	✔				✔	✔	1
Cookies Enabled	✓		✔								✔	1
JS detect: Flash Enabled	✓	✓	✓	✓	✔	✔					✔	
ActiveX + CLSIDs	✓	✓		✓	✔	✔						2
Date & Time		✔	✓	✔	✔							5
CPU		✔	✔		✔	✔				✔	✔	
System/User Language		✓	✔		✔						✔	2
OpenDatabase			✔		✔	✔					✔	
Canvas Fingerprinting					✔	✔		✔		✔	✔	
Mime-type Enumeration	✓			✓			✔	✔	✔		✔	
HTTP Proxy Detection			✓	✓								4
IndexedDB					✔	✔					✔	
Math Constants		✓			✔							5
Windows Registry		✓	✓									3
TCP/IP Parameters		✓	✓					✔				1
Google Gears Detection		✓										4
Flash Manufacturer				✓								4
MSIE Security Policy		✓										2
AJAX Implementation		✓										5
MSIE Product key			✓									3
Device Enumeration		✓										3
Device Identifiers			✓									3
IP address		✓						✔				1
HTML Body Behavior						✔						2
Battery					✔			✔			✔	
WebGLRenderingContext					✔			✔		✔	✔	

✓: Attribute previously reported in [13].
— : Attribute previously reported, but no longer present.

Table 1. Comparison of attributes used by various fingerprinting libraries.

Fingerprinters		Updated from
Pan	Panopticlick [5]	[13]
BC	BlueCava	[13]
IO	Iovation	[13]
TM	ThreatMetrix	[13]
Add	AddThis	**new**
FPjs	FingerPrintJS	**new**

Countermeasures	
FG	FireGloves [3]
Tor	Tor [4] Browser Bundle
PV	PriVaricator [12]
RAS	Random Agent Spoofer
FPB	**FingerPrint-Block**

[1] Property can be checked passively, i.e., no client-side technology required.
[2] Property specific to Internet Explorer.
[3] Property is determined using a Windows DLL created by the fingerprinting company.
[4] Out of scope – *FP-Block* only targets HTTP and Javascript layers.
[5] Blocking or spoofing this attribute would break or limit important functionality.

Table 2. Legend and footnotes for Table 1

Browser (Tor), PriVaricator [12] (PV), and Random Agent Spoofer[6] (RAS). For each, we determine which characteristics are considered. The results are listed on the right side of Table 1.

Not all characteristics are equal: some can be used by the fingerprinter without using JavaScript (e.g. HTTP headers), some are specific to Internet Explorer, others require additional client-side software (e.g. Windows DLL), etc. For the implementation, we do not focus on these characteristics. Furthermore, some characteristics are intrinsic to the communication or the user experience (HTTP accept header, date & time), and cannot be blocked or spoofed without adverse effects. Taken together, these considerations lead us to formulate the intended fingerprint surface of our tool *FP-Block* in the rightmost column of Table 1. Consequently, *FP-Block* only generates web identities which are distinct with respect to this fingerprint surface.

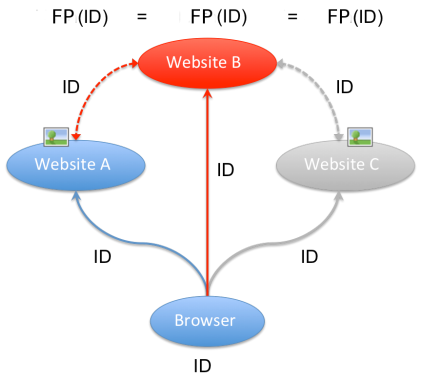
4 Design

Our goal is to prevent third-party fingerprint-based tracking. To this end, we generate distinct web identities (i.e., sets of fingerprintable characteristics) and then use these web identities in such a fashion as to be untraceable. More precisely, for each new site visited, we use a freshly generated web identity that is distinct from all previous web identities. This web identity is then used for all interactions due to this site, be they with the site itself or with any of its third parties.

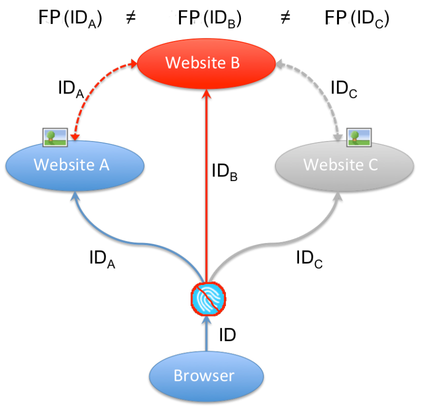
4.1 Balancing Usability vs. Privacy

The approach of separating web identities is not meant to interfere with regular tracking (tracking by the main site itself). It even allows third parties to

[6] https://github.com/jmealo/random-ua.js



a. Normal situation.



b. Separating web identities.

Fig. 1. Third-party fingerprinting.

track a user on a particular site – but it prevents third parties from linking that user to another user on a *different* site. Thus, this approach has the advantage that it does not affect local login processes, nor is it affected by such a process. Moreover, third parties that embed site-specific codes (e.g. of the form `http://facebook.com/FROM-SITE-A/`) are free to do so. We remark that the defensive paradox, i.e. defenses make the user stand out more, strongly impacts regular tracking. With regards to third-party tracking, however, there is a great difference. Our approach focuses on linkability between different sites, not on distinguishing users on one site. A user that stands out on one website is not necessarily the same person as a user that stands out on another website, even if both stand out due to the defensive paradox. Hence, third-party tracking is affected less severely by the defensive paradox.

The approach of separating web identities thus stays very close to a normal user experience. However, when visiting a different site, a different web identity is used, and the user cannot be tracked to this new site by third-party fingerprint-based tracking. Figure 1a depicts the normal functioning of the web: embedded third parties (shown by the dashed arrows) can fingerprint the site's visitors and match a fingerprint on site A to a fingerprint on site B. Figure 1b depicts our approach: each website visited sees another fingerprint. Consider that sites A and C embed a social media plugin of site B. Then when the user visits site A, the web identity as seen by B is ID_A. When the user visits site C, however, the web identity as determined by B is ID_C. Finally, if the user visits B directly, yet another web identity is used. This allows B to track the user locally on site A and on C, but not *from* site A *to* C.

4.2 Generating Web Identities

To prevent fingerprint-based tracking, we need to ensure that two distinct web identities are seen as different by a fingerprinter. This is not necessarily as straightforward as it may seem: computers and browsers are regularly updated (changing their characteristics), and fingerprinters need to account for this. Thus, merely ensuring that the set of characteristics of one web identity do not coincide with any other web identity is not sufficient. Moreover, we recall the fingerprinting countermeasure paradox: the use of a countermeasure impacts fingerprintable characteristics in such a way that the resulting set of characteristics is more unique (thus, more traceable) than if no countermeasure was used. An example of this is a client that claims to be an iPhone 2, that is capable of running Flash (real iPhones do not run Flash).

This leads to two design requirements: web identities must be "enough" different, and a generated web identity must be "consistent".

Ensuring sufficient difference. Updating a browser affects browser-specific attributes, updating the operating system affects OS-specific attributes, etc. To ensure a freshly generated web identity is sufficiently different from previously generated web identities, accounting for anticipated updates, we group the attributes into the following classes:

- Browser, e.g. user agent, browser name, vendor, accept-encoding.
- Language, e.g. language, system language, user language.
- OS/CPU, e.g. platform, cpu class, oscpu.
- Screen, e.g. width, height, color depth.
- Timezone. i.e. time zone offset.

Ideally, a freshly generated web identity is different in all classes to all previous web identities. This impacts the time required to generate a new web identity. For *FP-Block*, we chose to require every newly generated web identity to have at least two different attributes from at least two different classes. This allows a decent trade off between time needed to generate a web identity and uniqueness. In future versions, the generation algorithm could be optimised to require more differences.

Consistency of web identity. A randomly generated web identity is unlikely to be consistent, that is, have no contradictory attributes (such as Flash running on an iPhone). To ensure consistency, web identities need to be generated using a realistic distribution of attribute values. For instance, the chance that a Firefox user is on a Linux computer is greater than the chance that an Internet Explorer user is. In effect, the process to generate a consistent web identity can be modelled as a Markov chain. As usage of different browsers and operating systems varies over time, such a Markov chain needs to be updatable to remain current. To this end, we identified classes of states (operating system, processor, screen properties, etc). Any Markov chain for a web identity needs to contain these classes. Moreover, these can be ordered (operating system and processor

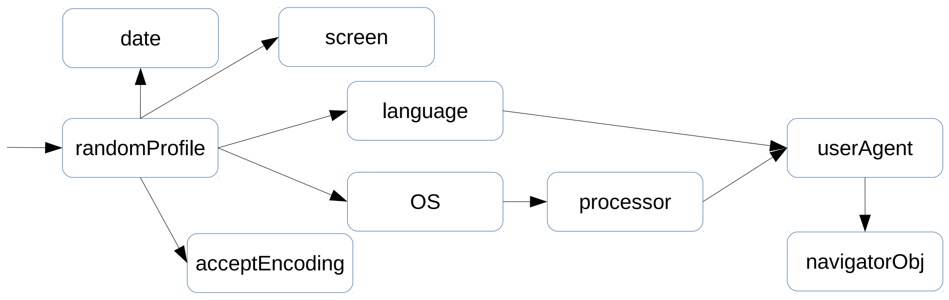


Fig. 2. Generic model of Markov chains for web identities.

are determined before the user agent is). In this way, we model a construction process for Markov chains generating web identities (see Fig. 2).

States and transition probabilities for Markov chains are derived from J. Mealo's data[7], based on actual usage statistics from Wikimedia. An excerpt of this data is shown in Table 3, and an example of how this translates into a Markov chain is shown in Fig. 3. In this figure, where no weights are indicated, the uniform distribution is implied. Remark that for windows operating systems, one of the attribute values for processor is the empty string ‘’.

	Win	Mac	Linux
chrome	.89	.09	.02
firefox	.83	.16	.01
opera	.91	.03	.06
safari	.04	.96	
iexplorer	1.00		

Platform	Processor string	probability
Linux	‘i686’:	.5
	‘x86_64’:	.5
Mac	‘Intel’:	.48
	‘PPC’:	.01
	‘U; Intel’:	.48
	‘U; PPC’:	.01
Windows	‘’:	$.\bar{3}$
	‘WOW64’:	$.\bar{3}$
	‘Win64; x64’:	$.\bar{3}$

Table 3. Example attribute distributions, due to J. Mealo

5 Development and Implementation

5.1 Development

We developed *FP-Block* as a Firefox plugin. This ensures that *FP-Block* is publicly available and easy to install for a large group of users. To limit the scope of

[7] https://github.com/jmealo/random-ua.js/commits/master/random_ua.js

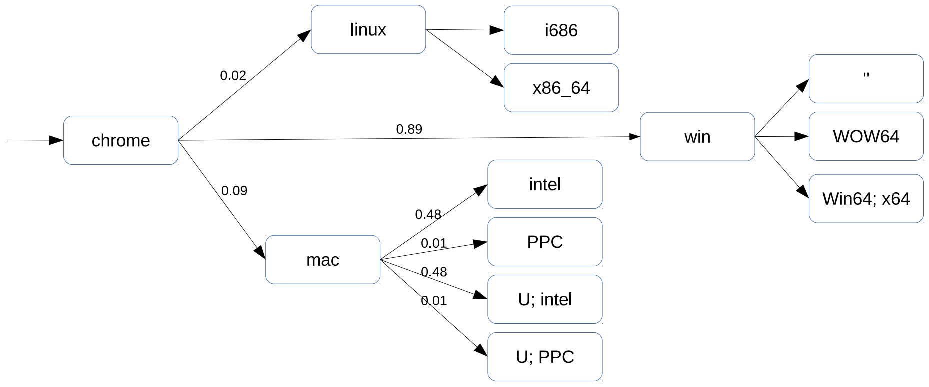


Fig. 3. Partial Markov chain for web identities for the Chrome profile.

the project, *FP-Block* focuses on two communication layers: JavaScript, illustrating application of our approach to active fingerprinting, and HTTP, illustrating application to passive fingerprinting. In principle, tracking at other layers was not considered. An exception to this was made for Flash: all commercial fingerprinters use Flash to fingerprint and store information. To be effective, the tool hides Flash (i.e., removed from detected plugins). Another exception is made for ActiveX. As *FP-Block* does not include a full ActiveX implementation, it cannot consistently pretend to be an Internet Explorer browser. Therefore, *FP-Block* never pretends to be Internet Explorer. Finally, *FP-Block* is designed for desktop browsers, and simulates only desktop browsers.

FP-Block is available from `http://satoss.uni.lu/software/fp-block/`, and is open source[8].

5.2 Implementation

FP-Block intercepts and modifies all outgoing requests and incoming responses. This is done by adding observers for the "topics" `http-on-modify-request` and `http-on-examine-{cached-}response`, respectively. For requests, first the existing web identity for the domain is retrieved, or, if none exists, a fresh web identity is generated and stored. Then HTTP headers like `Accept-Encoding` and `Accept-Language` are set according to the web identity. E-tag headers (which are intended to aid caching) can easily be exploited to track users, and are thus deleted (cf. Table 1). Finally, requests for social plugins that are explicitly blocked by user preference are cancelled. Currently, *FP-Block* can block the social plugins of Facebook, Twitter, Google Plus, LinkedIn, Tumblr, and Pinterest.

For incoming responses, the observer evaluates the fingerprint and constructs a

[8] Source available from GitHub repository FP-Block under the GPL v3 license.

Javascript script that enforces the fingerprint. Blocking access to blocked and spoofed attributes is handled by replacing the access functionality of Javascript using `navigator.__defineGetter__()`, for attributes such as `userAgent`, `appName`, `cookieEnabled`, `battery`, `geolocation`, `oscpu`, etc (cf. Table 1). Screen properties are protected analogously using `screen.__defineGetter__()`. Detection events are inserted in a similar way to detect when scripts use code that is typically used to fingerprint the user, such as enumerating mimetypes or plugins, using DOM storage, font detection, etc. Canvas fingerprinting is thwarted by adding random noise and the *FP-Block* logo to a random position on the canvas. This is then stored with the fingerprint to be reused in future visits. Finally, a few lines are added to remove the constructed script after execution, using `removeChild()` on the script's parent node. This ensures that the script cannot be found in the source by other scripts.

The thusly constructed script is then injected into the response at the very top, ensuring it is run before any other script sent by the server.

6 Experiments and Validation

We tested *FP-Block* in two different settings:

– *First-party fingerprinting*: the visited site uses fingerprinting, and
– *Third-party fingerprinting*: the visited site embeds a third-party fingerprinter.

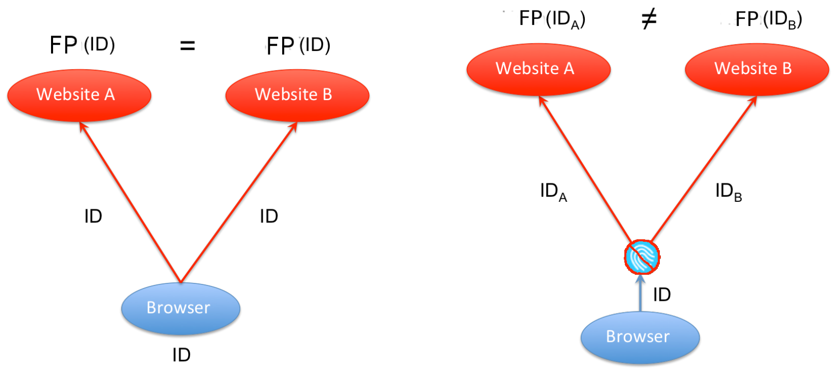
We ran three main variations of the tests:

1. without *FP-Block*.
   This is to verify that the test setup is functioning correctly.
2. with *FP-Block*,
   a. with the test site using an open source fingerprinting script, `FingerPrintJS`.
      This allows us to verify that *FP-Block* affects fingerprinting.
   b. with the test sites using any one of the four commercial fingerprinting scripts.
      This is to test the effectiveness of *FP-Block* against real-life fingerprinters.

We updated each fingerprinting script to add the fingerprint it computed to the test page. This allows us to execute the tests as described by simply opening the test sites in a browser (see e.g. Fig. 4).

Test 1: first-party fingerprinting. After successfully verifying that the test was set up correctly (Fig. 4a), we executed the test shown in Fig. 4b using FingerPrintJS. The result was that *FP-Block* correctly ensures that a browser visiting the two sites has a different fingerprint on each site.

Test 2: embedded fingerprinting. We created a test page on site A which embeds fingerprinting content from site B, and vice versa (cf. Fig. 1). Both test sites also ran the same fingerprint scripts locally. We then visited each page with and



a. Test case 1: no *FP-Block*. b. Test case 2: with *FP-Block*.

Fig. 4. First party fingerprinting tests.

without our tool. In both cases, we first tested this set up with the FingerPrintJS script.

Without *FP-Block*'s protection, the fingerprint is the same in all cases (cf. Fig. 1a). With *FP-Block*, however, this changes (Fig. 1b). The fingerprint script running on site A sees web identity ID_A if the user visits A, but if the user visits B (which embeds the script from A), the same fingerprint script sees web identity ID_B. The script returns different fingerprints for these web identities.

Testing against commercial fingerprinters. We then repeated tests, using the fingerprint scripts of the commercial fingerprinters (BlueCava, IOvation, ThreatMetrix, and AddThis). These scripts all compute a hash which seems to serve as the fingerprint. However, each of them also communicates the entire set of attributes and values used in this fingerprint back to the fingerprinter's servers. Our tests showed that, without *FP-Block*, the hash as computed by each script does not change – the behaviour is equivalent to that in Fig. 1a. With *FP-Block*, the hashes were different if they were computed by B's script embedded on site A, or by the same script of B computed when visiting site B. In short, the tests show that *FP-Block* successfully affects the fingerprint.

However, affecting the fingerprint does not necessarily mean that tracking is stopped. Given that commercial fingerprinters communicate the determined attribute values back to the fingerprinter, the fingerprinter may match such a set offline to previously seen sets. To test whether *FP-Block* is able to prevent such offline fingerprinting, we need to know the web identity which the fingerprinter attributes to us.

BlueCava provides such a service (the BlueCava advertising ID, available on BlueCava's "opt-out" page). Our final validation was to check our BlueCava advertising ID with and without *FP-Block*. The ID changed, which leads us to conclude that *FP-Block* successfully prevented BlueCava from tracking us. As can be seen in Table 1, BlueCava uses the most attributes for its fingerprint

– they are by far the most advanced fingerprinters listed. As such, we believe that success against BlueCava provides a good estimate of success against other offline fingerprint algorithms.

Update frequency of fingerprinters. Finally, we monitored the rate of change of fingerprinters listed in Table 1. We downloaded each fingerprinter's script once an hour from September 2014 until June 2015. In this period, Panopticlick's script did not change. The open source script FingerPrintJS changed once, which turned out to be support for detecting screen orientation.

With respect to the commercial fingerprinters: the scripts of BlueCava and AddThis have not changed since we began monitoring them. The scripts for IOvation and ThreatMetrix include time and date of the download, ensuring that every downloaded script is different. Since both scripts are heavily obfuscated, verifying that there are no changes other than embedded time and date is difficult. However, the file size of IOvation's script remained constant since September 2014. We take this as an indication that the script has not changed. Finally, ThreatMetrix' script started changing on 27 October, 2014, and still continues to evolve swiftly. An initial analysis revealed that there are large changes to the code base. Once ThreatMetrix seems to be more stable, we intend to re-analyse the new code.

Stability and compatibility testing. To determine the robustness of our plugin, we have been using the plugin in our main browser since June 2014. Where early versions occasionally crashed or gave unexpected results, since October 2014 only Javascript-heavy and Flash pages are noticeably affected. *FP-Block* blocks Flash to prevent Flash side channels, but Flash can be enabled in *FP-Block*'s menu. Javascript-heavy pages are slow to load. A refresh solves this.

Lastly, we tested how the plugin cooperates with several popular privacy-focused plugins: AdBlock Plus, Privacy Badger, Ghostery, and Disconnect. *FP-Block* perfectly cooperates alongside all in default settings. We note two caveats to this. First, Disconnect blocks social media plugins, similar to *FP-Block*. This functionality is implemented similarly in the two plugins. When both are running, social media plugins are blocked by both. Enabling a social media plugin thus requires enabling it in both plugins. Second, AdBlock Plus is a generic blocking plugin, into which blocklists can be loaded. There exist blocklists for social plugins. Loading such a blocklist inherently causes interference between AdBlock Plus and *FP-Block*.

7 Conclusions

Ubiquitous tracking on the web is a reality, enabled by the proliferation of embedded content and effective fingerprinting techniques. Currently available countermeasures against fingerprinting work by either blocking embedded content or faking a web client's characteristics. Not only does this break benign applications of fingerprinting (such as detecting session hijacking), but it also reduces the user experience. In order to achieve a more practical balance between privacy and usability, we have introduced the notion of a web identity that allows for re-identification within a web site, while it prevents tracking across websites. Such a web identity is a set of fingerprintable characteristics that can be tweaked on the user side.

We have developed a prototype web browser extension that supports the use and management of web identities. In order to design our tool's capabilities, we investigated the *fingerprint vector*, i.e., the set of characteristics used for fingerprinting, of the major fingerprint tools. This led to an up-to-date overview of four major fingerprinters' abilities.

The web identities generated by our tool are *distinct* and *consistent*. This means that two generated web identities are sufficiently different to prevent being linked by current fingerprint tools and that the attributes are being spoofed in such a way that their combination doesn't stand out. Consistency is achieved by implementing a Markov model for the generating of attribute values.

Fingerprint tools will be able to re-identify users even after minor changes or regular updates of their computing environment. Therefore, our tool should not generate web identities that can be linked in this way. Our current approach, consisting of classifying attributes, is a rather crude heuristic. We consider a refinement of this approach as interesting future work. Thereto, we propose to collect data on the evolution of client side attributes, in order to build a Markov model that will decide if two generated web identities are sufficiently distinct.

Finally, our prototype implementation only addresses the HTTP and JavaScript layers of communication. Given the focus of the current fingerprinting tools, this already provides a significant level of practical privacy. Nevertheless, the arms race between fingerprinters and anti-tracking tools will continue, so we consider extending the current fingerprint vector of our tool as an ongoing activity.

References

1. G. Acar, C. Eubank, S. Englehardt, M. Juárez, A. Narayanan, and C. Díaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proc. 21st ACM Conference on Computer and Communications Security (CCS'14)*, pages 674–689. ACM Press, 2014.
2. G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel. FPDetective: Dusting the web for fingerprinters. In *Proc. 20th ACM SIGSAC Conference on Computer and Communications Security (CCS'13)*, pages 1129–1140. ACM Press, 2013.
3. K. Boda, Á. Máté Földes, G. György Gulyás, and S. Imre. User tracking on the web via cross-browser fingerprinting. In *Proc. 16th Nordic Conference on Information Security Technology for Applications (NordSec'11)*, volume 7161 of *LNCS*, pages 31–46, Berlin, Heidelberg, 2011. Springer-Verlag.
4. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington, 2004.
5. P. Eckersley. How unique is your web browser? In *Proc. 10th Symposium on Privacy Enhancing Technologies (PETS'10)*, volume 6205 of *LNCS*, pages 1–18. Springer-Verlag, 2010.

6. T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.
7. B. Krishnamurthy and C.E. Wills. Generating a privacy footprint on the internet. In *Proc. 6th ACM SIGCOMM Conference on Internet measurement (ICM'06)*, pages 65–70. ACM Press, 2006.
8. J.C. Mitchell and J.R. Mayer. Third-party web tracking: Policy and technology. *Proc. IEEE Symposium on Security and Privacy (S&P'12)*, 0:413–427, 2012.
9. K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham. Fingerprinting information in JavaScript implementations. In *Proc. Web 2.0 Security & Privacy (W2SP'11)*. IEEE Computer Society, 2011.
10. K. Mowery and H. Shacham. Pixel Perfect: Fingerprinting canvas in HTML5. In *Proc. Web 2.0 Security & Privacy (W2SP'12)*. IEEE Computer Society, 2012.
11. M. Mulazzani, P. Reschl, M. Huber, M. Leithner, S. Schrittwieser, and E.R. Weippl. Fast and reliable browser identification with javascript engine fingerprinting. In *Proc. Web 2.0 Security & Privacy (W2SP'13)*, 5 2013.
12. N. Nikiforakis, W. Joosen, and B. Livshits. PriVaricator: Deceiving fingerprinters with little white lies. Technical Report MSR-TR-2014-26, Microsoft Research, February 2014.
13. N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Proc. 34th IEEE Symposium on Security and Privacy (S&P'13)*, pages 541–555. IEEE Computer Society, 2013.
14. F. Roesner, T. Kohno, and D. Wetherall. Detecting and defending against third-party tracking on the web. In *Proc. 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, pages 155–168. USENIX, 2012.
15. A. Roosendaal. We are all connected to facebook ... by facebook! In *European Data Protection: In Good Health*, pages 3–19. Springer-Verlag, 2012.
16. T. Unger, M. Mulazzani, D. Fruhwirt, M. Huber, S. Schrittwieser, and E.R. Weippl. SHPF: Enhancing http(s) session security with browser fingerprinting. In *Proc. Eighth International Conference on Availability, Reliability and Security (ARES'13)*, pages 255–261. IEEE Computer Society, 2013.
17. 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 *Proc. 19th Annual Network & Distributed System Security Symposium (NDSS'12)*. The Internet Society, 2012.