GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Hot Topics in Computer Security

Cookieless Monster: Exploring the Ecosystem of Web-based Device Fingerprinting

21.07.2013

Frederick Beyer (Mat.-Nr. 20837680)

Table of Contents

1 – Introduction 3

1.1 – Fingerprinting and Third-Party Tracking 3

1.2 – The Cookieless Monster 3

2 – Commercial Fingerprinting 4

2.1 – Introduction: Vendors and use cases 4

2.2 – Overview: Methods and features 5

2.2.1 – Basic fingerprinting 6

2.2.2 – Vendor-specific fingerprinting 7

2.2.3 – Detection of fonts 7

2.2.4 – Detection of browser-defined HTTP proxies 7

2.2.5 – System fingerprinting plugins 8

2.2.6 – Other 8

2.3 – Implementation: Delivery mechanisms and scenarios 9

3 – Adoption of Fingerprinting 9

3.1 – Fingerprinting on the popular web 9

3.2 – Fingerprinting on other sites 10

3.3 – Discussion 10

4 – Fingerprinting the behavior of special objects 10

4.1 – Experimental fingerprinting setup 11

4.2 – Results and Evaluation 11

5 – Countermeasures and fingerprinting resistance 12

5.1 – Reducing the fingerprintable surface 13

5.2 – User-agent spoofing 13

5.3 –Content blocking 14

6 –Discussion and conclusion 16

References 17

1 – Introduction

1.1 – Fingerprinting and Third-Party Tracking

Online privacy is a perpetually relevant topic, and has been a concern of users for a long time (cf. e. g. Bau et al. 2013 p. 1 for a list of surveys indicating this). However, as important as the Internet's relative anonymity is in public perception (and arguably use) of the Web, this privacy is arguably not "by design". Instead, it has been argued, it is something of an *emergent* feature; the initial architecture placed a lower priority on accountability than e. g. disruption tolerance, which led to a somewhat vague notion of identity and thus relative anonymity. (cf. e. g. Mayer 2010, p. 12ff.)

User privacy thus is somewhat "frail", given that its protection is not in fact a core feature, and there are (and have been) many attempts to partially remove this relative anonymity. There arguably are benign reasons to attempt this and to e. g. tie users to their browsing history. For instance, websites which offer paid-for subscriptions have a legit interest in recognizing and preventing account-sharing; similarly, advertisers may wish to better tailor their offers to the visitor in question, which requires the ability to identify and track specific users. Historically, tracking and identification have relied on client-side identifiers such as cookies. This method is far from fool-proof, and users can opt out easily by selectively rejecting cookies; this is beneficial from a privacy perspective, but also vastly reduces the method's usefulness in e. g. fraud prevention. (cf. e. g. Nikiforakis et al. 2013, p. 1f., Bau et al. 2013, p. 1) Therefore, different methods have been developed.

A current topic of concern in this context is that of *web-based fingerprinting* – and, based on this, *third-party tracking*. Both of these expressions are somewhat vague. For the context of this paper, the term "web-based fingerprint" will be used to mean *a collection of attributes a browser makes available to a website that can possibly identify the browser or device*, with "fingerprinting" being the collection and condensation of these characteristics into an identifier. A "fingerprintable feature", then, is a characteristic that can possibly be used in fingerprinting, and a "fingerprinting method" an approach to obtain such information. "(Cookieless) third-party tracking" is a potential application of fingerprinting and will be explained in more depth in section 2.1. (cf. e. g. Eckersley 2010 p. 4ff., Bau et al. 2013, p. 1)

1.2 – The Cookieless Monster

A few papers have been written on this topic in recent years. Among them is Nikiforakis et al.'s 2013 paper "Cookieless Monster: Exploring the Ecosystem of Web-based Device Fingerprinting", which will this text will attempt to describe and present.

The authors aim to provide an overview over (and classification of) different fingerprintable features. Their analysis was done in four steps. First, the authors analyze fingerprinting scripts used by three major commercial vendors (BlueCava, Iovation, ThreatMetrix), chosen partially based on previous research. Nikiforakis et al. then compare them to one another and to Eckersley's Panopticlick (an "open-source" implementation of browser fingerprinting). After this, Nikkiforakis et al. describe a fairly effective experimental fingerprinting setup based on the behavior of the Java-Script accessible browser objects `navigator` and `screen`, which contain information about the browser itself (e. g.

name and version) and the user's screen (e. g. width and height). Finally, they investigate potential countermeasures or ways fingerprinting could be prevented; their analysis mostly constrains itself to user-agent-spoofing browser extensions, which they find to be fairly ineffective. The authors conclude that privacy is currently on the losing side, as fingerprinting can be problematic from a privacy perspective (being that it can be used to strip anonymity from website visitors if they leave a sufficiently unique fingerprint) and can be hard to defend against. (cf. Nikiforakis et al. 2013, p. 1ff.)

As the main goal of this text is to present the findings of Nikkiforakis et al. and sketch the greater context in which they can be placed, it will roughly follow this structure. However, in the interests of a broader perspective, this text will occasionally stray a little, and it will not be followed rigidly.

Section 2 is thus dedicated to an initial rough sketch of the field. It contains a broad description of the above-mentioned commercial vendors and the purposes of fingerprinting, followed by an exploration of fingerprinting methods or *possible features* that could form part of a fingerprint. Section 3 considers the *actual use* of fingerprinting on the web as found by the authors of the paper, aiming to answer the question of how widespread it really is. Section 4 presents a condensed explanation of the aforementioned experimental fingerprinting setup. In section 5, possible countermeasures will be investigated; here, this text deviates slightly from the original paper's structure, as Nikiforakis et al. only investigate user-agent spoofing browser extensions. Finally, the results will be discussed.

2 – Commercial Fingerprinting

2.1 – Introduction: Vendors and use cases

As hinted above, there are several possible uses for fingerprinting, both benign and malicious. Interestingly, the three companies mentioned by Nikiforakis et al. cover a large range of these possible uses between them; therefore, the topics will be introduced together.

The first company mentioned in the text is *BlueCava*, which markets itself mainly to advertisers. BlueCava promises to be able to recognize and track visitors, possibly across multiple devices, to supply information about them, and to enable personalized offers. (cf. BlueCava website) Fingerprinting techniques can be used in this context to *identify* people and *track* them across multiple websites and sessions (without needing a client-side identifier), thereby creating a profile of a user's browsing habits, which can then be used to tailor content (such as advertising or special offers) to the user. (cf. Nikiforakis et al. 2013 p. 1f.)

The second company examined by Nikiforakis et al. is *Iovation*, which also describes its services as "device identification", albeit with a wholly different motivation – *fraud prevention*. This is purportedly achieved by with a "device reputation" system which calculates a risk score of sorts based on user-configurable rules and a wide range of possible metrics, including device characteristics and configuration. Additionally, according to the company, devices which have been in the past used to engage in fraudulent activities can be recognized. (cf. Iovation website) This, too, requires the use of fingerprinting techniques – both for information retrieval and for device identification, neither of

which would be possible with “traditional” client-side identifiers. For instance, fraudsters can hardly be expected to tolerate the presence of a cookie on their device that identifies them as such.

Finally, there is *ThreatMetrix*. This company, too, offers services belonging to the field of “cybercrime prevention” based on device identification, and claims to provide its customers a way to recognize and prevent identity theft. In their material, ThreatMetrix directly addresses the obsolescence of “traditional device identification” – IPs change (and can be cloaked via proxy), cookies altered or deleted. The company therefore offers “cookieless device identification “, which again relies on fingerprinting techniques. (cf. ThreatMetrix 2013, p. 1-8)

Finally, as “baseline” for the comparison, the authors used Eckersley’s *Panopticlick*, which is an “open” implementation of fingerprinting techniques created for research purposes. This may also qualify as a possible use of fingerprinting (cf. Eckersley 2010, p 1ff., https://panopticlick.eff.org/)

There are other possible use cases for fingerprinting techniques not represented among the four systems mentioned above, such as preventing visitors from voting multiple times in online polls (where IP-based systems can be bypassed via proxies and cookies simply rejected). Not all potential uses of fingerprinting techniques are benign, however. A malicious example would be the use of fingerprinting techniques in the course of *drive-by download attacks*. To briefly summarize the issue, the use of fingerprinting techniques in this context allows attackers to deliver “custom-tailored” exploits. By identifying a visitor’s browser, extension and operating system, attack (and delivered malware) can be configured as needed to increase the odds of a successful infection. (cf. e. g. Nikiforakis et al. 2013 p. 2, 13; Cova et al. 2010 p. 10)

However, even the seemingly benign applications are questionable from a privacy perspective. What is especially problematic in this context is that there is no easy (and enforceable) option for users to “opt out” of being fingerprinted and, potentially, tracked. This will be discussed in more depth in section 5. However, before the issue of countermeasures can be addressed, it is important to understand how fingerprinting (potentially) works, and why it is so difficult to prevent.

2.2 – Overview: Methods and features

Fingerprinting methods have a surprisingly long history, going back to at least the early 2000s; for instance, Veysset et al. published a paper on comprehensive “Remote Operating System Fingerprinting” as early as 2002 (cf. Veysset et al. 2002 p. 1 ff.). Similarly, there is a large variety of fingerprinting techniques, not all of which perform equally well. This text will only describe techniques that work through *the browser*, and focus (but not exclusively) on those mentioned by Nikiforakis et al.

There are many different possible approaches to web- or browser-fingerprinting. In many cases, methods or features can be combined to create both a “broad” and “deep” fingerprint – one which covers a wide variety of features both of the browser and the underlying (or surrounding) system. Nikiforakis et al. roughly grouped the features into five layers ranging from “browser customizations” to “hardware & network”. (cf. Nikiforakis et al. 2013 p. 3)

In the initial step of their analysis, Nikiforakis et al. obtained and analyzed fingerprinting scripts used by the four vendors mentioned above. Their results are given below. The table also illustrates this concept of a “layered fingerprint”, as well as which fingerprinting providers use which features and

which method is used to retrieve the information. Features in *italics* are those marked by the authors as being "sufficiently new or extended" compared to Panopticlick. The method used to obtain the information is also given, with JS standing for JavaScript and SFP for a system-fingerprinting plugin (see section 2.2.5). (cf. Nikiforakis et al. 2013 p. 2ff.)

	Panopticlick	**BlueCava**	**Iovation ReputationManager**	**ThreatMetrix**
Browser customizations	Plugin enumeration (JS) Mime-type enumeration (JS) ActiveX + 8 CLSIDs (JS)	Plugin enumeration (JS) *ActiveX + 53 CLSIDs (JS)* *Google Gears detect (JS)*		Plugin enumeration (JS) Mime-types (JS) ActiveX + 6 CLSIDs (JS) Flash manufacturer (Flash)
Browser user configurations	Cookies enabled? (HTTP) Timezone (JS) Flash enabled? (JS)	*Language (JS)* Timezone (JS) Flash enabled? (JS) *Do-Not-Track choice (JS)* *MSIE Security Policy (JS)*	*Language (HTTP, JS)* Timezone (JS) Flash enabled? (JS) *Date & time (JS)* *Proxy detection (Flash)*	*Browser language (Flash)* Timezone (JS, Flash) Flash enabled? (JS) *Proxy detection (Flash)*
Browser family and version	User-agent (HTTP) ACCEPT-Header (HTTP) Partial supercookie test (JS)	*User-agent (JS)* *Math constants (JS)* *AJAX implementation (JS)*	*User-agent (HTTP, JS)*	*User-agent (JS)*
OS and applications	User-agent (HTTP) Font detection (Flash, Java)	User-agent (JS) *Font detection (Flash, JS)* *Windows Registry (SFP)*	User-agent (HTTP, JS) *Windows Registry(SFP)* *MSIE Product key (SFP)*	User-agent (JS) Font detection (Flash) *OS & kernel version (Flash)*
Hardware and network	Screen resolution (JS)	Screen resolution (JS) *Driver enumeration (SFP)* *IP Address (HTTP)* *TCP/IP parameters (SFP)*	Screen resolution (JS) *Device identifiers (SFP)* *TCP/IP parameters (SFP)*	*Screen resolution (JS, Flash)*

Taxonomy of features used by the studied fingerprinting providers (cf. Nikiforakis et al. 2013 p. 3)

The following text, which will describe the authors' findings, will (very) roughly follow this structure, moving from the top to the bottom layer.

2.2.1 – Basic fingerprinting

The most important feature obtained (potentially) directly from the browser is the user-agent string (HTTP header), which directly states the operating system and web browser are being used. However, this information *alone* is usually not enough to identify a user with sufficient certainty, and it can be "faked" relatively easily (see section 5), which would make it unwise to rely solely on the user-agent string. However, it forms an important part in many fingerprinting scripts. (cf. Eckersley 2010b)

As the table above illustrates, both JavaScript and Adobe Flash are used extensively in the context of fingerprinting, as they offer fairly rich APIs through which information can be obtained. This pertains to all layers, but especially those where features are not "normally" surrendered by the browser. For instance, the list of fonts installed on the system can be obtained via Flash or JavaScript; this will be described in section 2.2.2 below. Interestingly, Java plugins were used by Panopticlick but none of the three commercial vendors, which the authors speculate is because they are unlikely to be executed (due to reputation as security threat). Furthermore, most of the fingerprinting vendors attempt to enumerate browser plugins. Internet Explorer is less forthcoming about this information; in

this case, fingerprinting scripts have to fall back on testing for plugins on a precomputed "list" (cf. Nikiforakis et al. 2013 p. 3)

2.2.2 – Vendor-specific fingerprinting

Another interesting finding was that fingerprinting scripts frequently target "vendor-specific features", attempting to access e. g. browser-specific properties of the `navigator` object (such as Internet Explorer's `navigator.systemLanguage`). This is rather unsurprising, as it allows a fingerprinting script to identify both the identity of a browser (see section 5) as well as potentially its version (see section 3). Furthermore, the values of these properties are themselves fingerprintable features, and thus valuable information. The fingerprinting scripts thus do *not* attempt to be "universal" but make extensive use of vendor-specific features (cf. Nikiforakis et al. 2013 p. 3f.).

2.2.3 – Detection of fonts

Users may have installed additional fonts on their system, or may be using operating systems with differing "default sets" of fonts. The font list on a device, therefore, can also be regarded as a fingerprintable feature. The authors describe two ways in which a list of fonts installed on a system can be retrieved: *Plugin-based detection* and *side-channel inference*.

Plugin-based detection is a straightforward font-detection method based on Adobe Flash. ActionScript provides an API that makes an ordered list of fonts available, which can simply be accessed and used as potentially highly informative feature for fingerprinting. If a user has sufficiently unusual fonts installed, this alone may be able to identify their device. (cf. Nikiforakis et al. 2013 p. 4.)

Side-channel inference is a less direct JavaScript-based method which relies, essentially, on a brute force attack. First, the script creates a text element (with previously specified font size and content) using the `sans` font family. This element's size (`offsetWidth`, `offsetHeight`) is then measured, and the values are stored. Then, the script iterates over a (potentially large) list of font names, and for each attempts to create an element containing the previously specified string at the set font size. The resulting element is again measured. If the values returned are *equal* to those of the "sans element", the script assumes that the font does not exist on the system (as the browser fell back on sans) and records this result; if the values are different, the font is assumed to exist. This method is less useful than plugin-based detection as it does not return an ordered list and cannot detect rare fonts, but it is nonetheless useful as a fallback. (cf. Nikiforakis et al. 2013 p. 4.)

2.2.4 – Detection of browser-defined HTTP proxies

A "traditional" way to identify devices is via IP address. This may not always be feasible, as devices may change IPs over time (dynamic IP); additionally, a user may make use of an anonymizer / proxy to hide their true IP address. However, knowing a user's IP may be highly valuable in the course of fingerprinting – IPs generally do not change *rapidly*, which would simplify tracking a user over at least over the course of a single session, and they may offer clues as to a user's geographic location. Fingerprinting (and tracking) companies therefore have a definite interest in being able to know a user's IP address even (or especially) if they take steps to conceal it (cf. Nikiforakis et al. 2013 p. 4f.).

Nikiforakis et al. found that this may, in fact, be possible. Flash applications are capable of *contacting remote hosts directly, bypassing browser-set HTTP proxies* and thereby leaking the user's true address to the remote host. In fact, if this method is used, a fingerprinter gains *more* information about the visitor than it normally would have – such as the fact that the user is trying to conceal their IP. This result can be seen as encouraging for the fingerprinting (and tracking) companies but is highly worrying from the user's point of view, being an invasion of privacy (as it bypasses active measures taken by a user to conceal their identity). (cf. Nikiforakis et al. 2013 p. 4-5.)

2.2.5 – System fingerprinting plugins

The fingerprintable features examined so far can be used to form a relatively descriptive fingerprint, but they remain relatively "superficial", as they primarily describe the browser and only to some extent its environment. Nikiforakis et al. found that some vendors (BlueCava and Iovation) attempted to gain additional information about "deeper" layers of the user's system. This was achieved through special plugins, which were distributed with (and able to invoke) DLLs that function essentially as native fingerprinting libraries. These libraries have access to a great amount of information about the user's system, and are able to reveal information belonging to layers "below" the user's browser – including but not limited to hardware identifiers, machine name, Windows installation date and Digital Product Id, installed system drivers and so forth. While this highly intrusive method requires the user to "opt-in" at some point (as the plugins have to be installed first), it is possible to bundle them with "desired" software (as has been done with e. g. browser toolbars in the past) and thereby place them onto a user's system. Nikiforakis et al. furthermore found that the plugins may attempt to conceal their purpose (by identifying themselves as "identity shields" or similar, which is only technically true), casting this practice in a very questionable light. (cf. Nikiforakis et al. 2013 p. 5) However, the information obtained through these plugins is highly machine-specific (possibly unique) and likely to change frequently, and is therefore extremely useful for fingerprinting.

2.2.6 – Other

While Nikiforakis et al. provide a fairly complete description of state-of-the-art fingerprinting, features and methods exist that have not been described in-depth in the paper (as they were not implemented in the scripts used by Panopticlick, BlueCava, Iovation or ThreatMetrix). As the authors acknowledge, their analysis was mostly manual and very time-consuming; "proprietary" features used by different vendors might have been missed by their study. (cf. Nikiforakis et al. 2013 p. 6)

Some alternative methods that have been proposed in the research community are impractical or intrusive (such as fingerprinting JavaScript engines through benchmark execution time; cf. Nikiforakis et al. 2013, p. 6) while others are promising, but not yet widely usable. An example for the latter would be Mowery/Schacham's HTLM5 `<canvas>`-tag based fingerprinting system. This approach works by rendering text or images to a `<canvas>` element, then reading the output back in pixel-wise. The authors obtained up to 5.73 bits of entropy through this method, although the participants in the study showed little variation in browser and OS. Nikiforakis et al. dismiss the method as it relies on features not present in older browsers, but it does provide an additional source of entropy orthogonal to existing ones with little overhead, and may see use in the future, especially in environments where Flash is unavailable. (cf. Mowery/Shacham 2012 p. 1-10).

2.3 – Implementation: Delivery mechanisms and scenarios

In addition to the information obtained by the scripts, Nikiforakis et al. also investigated *how* these scripts are deployed. This section will exclusively discuss delivery mechanisms in the context of *third*-party fingerprinting; in cases where the fingerprinting is done *directly* (e. g. in case of experimental setups such as Panopticlick), the mechanisms will obviously be different.

The authors distinguish between two basic scenarios for third-party fingerprinting.

In the first scenario, the fingerprinting is done entirely by a third party – the fingerprinting code was delivered by an advertising syndicator and the fingerprint transmitted back to it, without the first party being involved in the process; this method is used e. g. to combat click-fraud. In fact, the first-party website needs not even be aware of the fingerprinting. (cf. Nikiforakis et al. 2013 p. 5f.)

In the second scenario, the first party is the one requesting and possibly receiving the fingerprint; the fingerprint may, however, be encrypted, so that it has to be submitted back to the vendor for analysis. The vendor still receives the fingerprint in this case, allowing the company to check it against its list of profiles. Thus, even if the first party only requests a fingerprint for purposes of fraud detection, it may eventually still be used in third-party tracking (cf. Nikiforakis et al. 2013 p. 5f.).

3 – Adoption of Fingerprinting

Having created a structured overview of the features the four systems used in fingerprinting and of delivery mechanisms used to deploy them, the authors move on to investigate the *adoption* of fingerprinting. This issue is important from a "privacy-protection" point of view – is fingerprinting only a theoretical possibility, or is it an emerging threat?

3.1 – Fingerprinting on the popular web

First, Nikiforakis et al. attempted to quantify the use of fingerprinting on "popular" websites. As preparation, they retrieved a list of the 10,000 most popular websites from the service *Alexa* and classified them according to TrendMicro's domain categorization service. Then, up to 20 pages were crawled for each in an attempt to discover which of these websites make use of the fingerprinting scripts offered by the three companies mentioned above.

The result was that fingerprinting was somewhat rare but not unknown, with 0.4% (40) websites having been found to make use of a fingerprinting script supplied by BlueCava, Iovation or ThreatMetrix. As this method is prone to false negatives, this number is a lower bound; it is possible that the websites made use of fingerprinting scripts of *another* vendor, or simply did not perform fingerprinting on those pages that were crawled. The most popular categories of websites performing fingerprinting were "pornography" (15% / 6) and "personals/dating" (12.5% / 5); the authors speculate that websites in the former category are trying to prevent sharing paid memberships, while the latter are trying to prevent malicious users from creating multiple profiles. (cf. Nikiforakis et al. 2013 p. 6)

3.2 – Fingerprinting on other sites

In the next step of their analysis, the authors of the *Cookieless Monster* paper attempted to investigate the use of fingerprinting scripts on less popular websites; specifically, to find out which *types* of websites make use of fingerprinting. This was done by submitting the websites on a pre-existing list (3,804 domains of sites confirmed to request fingerprinting scripts) to TrendMicro's and McAfee's domain categorization websites and recording the results. If a website was flagged as malicious by one service but not the other, it was assumed to be not malicious; if the two services disagreed on other aspects, McAfee's assignment was given precedence. (cf. Nikiforakis et al. 2013 p. 6f.)

The results partially lined up with previous findings. In descending order of popularity, the most frequently-occurring categories in the data set were "Spam", "Malicious Sites", "Adult / Mature content", "Computers / Internet", "Dating / Personals", "Entertainment", "Business / Economy", "Internet Services", "Travel" and "Shopping". Many of these websites operate on a subscription model (particularly "Adult / Mature" and "Dating / Personals") and thus have a vested interest in preventing practices such as account sharing, which matches the results found in section 3.1 above.

On, the other hand, the fact that the top two categories were *malicious* seems somewhat surprising, and would speak ill of fingerprinting. In fact, more than 34% of the sites in the list were classified as "spam", with "malicious sites" accounting for another 4%. However, it should be noted that the data set was obtained in the context of *detecting drive-by download attacks* through dynamic JavaScript analysis and machine learning. (cf. Cova et al. 2010 p. 1ff.) Such scripts may make use of fingerprinting, as mentioned above, and therefore the presence of fingerprinting code may be a useful feature in identifying possibly malicious scripts. Because of the context in which the list was obtained, it is somewhat skewed towards maliciousness, as Nikiforakis et al. note themselves. The results from the second study should therefore be taken with a grain of salt. (cf. Nikiforakis et al. 2013 p. 6f.)

3.3 – Discussion

Both studies were somewhat limited. To recap briefly: The first study only provides a *lower bound* for the adoption of fingerprinting, and the second study was based on a data set somewhat predisposed towards malicious content. Nonetheless, they offer some vital clues regarding the adoption of fingerprinting – it is no niche application, and vendors appear to sometimes cooperate with questionable partners, suggesting that it may in fact qualify as emerging threat to privacy.

4 – Fingerprinting the behavior of special objects

Having completed their initial survey of fingerprinting as it exists on the web, Nikiforakis et al. went on to design a fingerprinting setup of their own. The authors' fingerprinting setup focuses on two browser-populated JavaScript objects – `navigator` and `screen` – which contain information highly relevant for fingerprinting purposes, such as the name and version of the browser and width and height and the user's screen, respectively. Their setup is intended to demonstrate that vendors do not need to rely on the "goodwill" of browsers to surrender true information, as it is surprisingly difficult to achieve full coverage of all potential leaks (see section 5). (cf. Nikiforakis et al. 2013 p. 7f.)

4.1 – Experimental fingerprinting setup

Their approach is fairly simple, performing a series of simple operations on both special objects and recording the results at each step (by requesting a list of all properties of each object, Base64-encoding the result and storing it). Specifically, the following operations were performed (cf. Nikiforakis et al. 2013 p. 7f.)

- Enumeration of both objects, requesting a list of all properties of each.
- Re-enumeration of the `navigator` object to test if property list is ordered
- Creation, population and enumeration of a custom object as test to see if the "special" objects behave differently from "normal" JavaScript-created objects.
- Attempted deletion of a property for each of the `navigator`, `screen` and custom objects.
- Attempted restoration of the deleted property.
- Attempted modification of existing properties of the `navigator` and `screen` objects.
- Attempted flagging of an existing property of the special objects as non-enumerable.
- Attempted deletion of the `navigator` and `screen` objects.
- Attempted assignation of a new object to the `navigator` and `screen` variable names.

4.2 – Results and Evaluation

Nikiforakis et al. then conducted a small-scale study (one week, 68 points of data) as initial test of the fingerprinting setup. The results were fairly impressive, and the authors found several novel ways of distinguishing between browser families and versions, based on property enumeration, the presence (or absence) of unique features, mutability of special objects and functionality evolution. (cf. Nikiforakis et al. 2013 p. 8)

The *property-enumeration* of special browser objects is a fingerprintable feature, differing between browser families, versions and (in case of Chrome) deployments. This applies to both the properties returned and, in some cases, the order of the list. `navigator` objects with fewer properties consistently belonged to older browser versions, and all browsers except Chrome maintained the ordering of navigator elements between versions (meaning that the relative order of older properties did not change when new properties were added – that is, if the newly-added features are removed, the list remains the same). In Chrome, the order seemed to be random, changing between subsequent versions of the browser and between deployments of the same version on different systems. While less straightforward to exploit, this, too, forms an "opening" for fingerprinting. (cf. Nikiforakis et al. 2013 p. 8f.)

In all browsers, certain *unique features* were present – all browser families examined had at least two features not shared by any other browser, such as `screen.mozBrightness` for Mozilla Firefox or `navigator.msDoNotTrack` for Microsoft Internet Explorer. This allows a fingerprinting script to identify a browser's family with a high degree of certainty, as well as its version (as these features are gradually added over time), even if a user takes steps to conceal these. This will be discussed in more detail in section 5. (cf. Nikiforakis et al. 2013 p. 9f.)

The special objects were treated differently by different browsers / browser families, especially regarding their *mutability*. For example, Google Chrome allowed a script to delete a property from the `navigator` object. Google Chrome and Opera allowed a script to modify a value of the `navigator` object or reassign the names `navigator` and `screen` to other objects, requests which failed

silently for Mozilla Firefox and Microsoft Internet Explorer. Mozilla Firefox, finally, behaved as if a property of the `navigator` object was actually *deleted* when a script was made non-enumerable. This, too, allows a script to gain information about a browser's "true" family if a user attempts to hide it. (cf. Nikiforakis et al. 2013 p. 10.)

As hinted above, functionality *evolves* over time as vendors introduce new features, which also allows a script to gain information about a browser's version (and family). The presence (or absence) of certain features therefore qualifies as a fingerprintable characteristic, even if the features in question are not vendor-specific. The authors investigated Google Chrome specifically and found that one can distinguish between all major versions (and sometimes minor versions) by checking simply for specific features. (cf. Nikiforakis et al. 2013 p. 10f.)

Finally, the browsers behaved subtly different in various *miscellaneous* ways such returning subtly different results when `toString()` is called for the examined objects. For instance, the result of invoking the `toString()` method on the `navigator.plugins` object, browsers may report anything ranging from "`[object]`" (Internet Explorer)" over "`[object PluginArray]`" (Mozilla Firefox / Opera) to "`[object DOMPluginArray]`" (Google Chrome). (cf. Nikiforakis et al. 2013 p. 11.)

It is important to view these results in context. The experimental fingerprinting setup used by Nikiforakis et al. is only *one* possible variant. Other implementations of the same concept may fingerprint different features, or in different ways, or include (or lack) "fallbacks" to address users' privacy-protecting measures, and thus yield different results.

However, to briefly summarize the results: Fingerprinting the behavior of special objects can identify a browser's family and version with a surprisingly high degree of certainty. Although it requires some effort to construct the "profiles" in the first place, the resulting method is both effective and efficient, and can be used together with more traditional methods of fingerprinting. It is especially useful in cases where users may attempt to disguise their "true" fingerprints, which will be discussed below.

5 – Countermeasures and fingerprinting resistance

Sections 2 and 4 have shown that fingerprinting is theoretically feasible and offers many possibilities to accurately identify visitors. Section 3 has shown that, while fingerprinting appears to not yet be widespread, it has started to appear "in the wild"; this trend can reasonably be expected to continue. This trend is worrying from a privacy perspective, as fingerprinting can easily be done without a user's knowledge or consent. With traditional cookie-based device identification, it is relatively easy for users to "opt out" (even if the vendor may not desire it) simply by selectively rejecting (and/or deleting) cookies as needed. The proposed (and partially implemented) "Do Not Track" header may appear as such an option at first glance, but actually does little to change this. Users will still be fingerprinted and are forced to rely on the "goodwill" of third parties regarding tracking. In the case of a "rogue" company, all the DNT header achieves is adding another field to the fingerprint. (cf. Nikiforakis et al. 2013 p. 2).

As fingerprint-based device recognition does not require client-side identifiers, it cannot be prevented quite this easily. This section is dedicated to a brief investigation of possible *countermeasures* and *sources of* resistance against fingerprinting.

5.1 – Reducing the fingerprintable surface

The basic problem when "defending" oneself against fingerprinting, as has been hinted above, is not so much preventing fingerprinting outright as making sure that the fingerprint *does not contain enough information* to identify a visitor with a sufficiently high degree of certainty. Eckersley estimated that roughly 15-20 bits of identifying information will be needed to uniquely identify a particular browser; if one was able to reduce the amount of information leaked sufficiently far below this number, identification via fingerprint would become infeasible (cf. Eckersley 2010, p. 3).

One obvious way in which this could be achieved would be through browser vendor action as suggested by Nikiforakis et al. If browsers were "equalized" in terms of behavior (e. g. of special objects as described above), the fingerprintable surface could be reduced. If done on the browser-level, the limitations of plugins/extensions do not apply. (cf. Nikiforakis et al. 2013, p. 14) However, it is questionable if this is really desirable from the browser vendors' point of view. Not only does it require a massive amount of effort and coordination, but it also does not grant them any competitive advantage – a "fingerprint-safe" browser would *by definition* not be distinguishable (in terms of behavior or performance) from any other "fingerprint-safe" browser. Similar arguments apply for optional "fingerprint-safe modes"; if only one browser were to offer this, it would be no advantage, for the same reasons. This, therefore, is unlikely to manifest. The rest of this section is therefore dedicated to investigating measures individual *users* can take to render themselves less "fingerprintable" (and thus "trackable"), whatever their intention.

In Eckersley's 2010 study, four groups were found to be relatively resistant to fingerprinting: Those who use the browser extension TorButton, those who block JavaScript, certain types of smartphone (as there is no such wide variety of plugins) and machines with synchronized ("cloned") software installations (e. g. corporate environments). The fourth group only partially qualifies as being resistant – while this setup may protect against software-based fingerprints, it does not necessarily prevent *hardware*-based fingerprinting (cf. Eckersley 2010, p. 3-4/16).

Nikiforakis et al. found somewhat different results, particularly regarding the value of extensions such as TorButton. This will be described and discussed in section 5.2, and further illustrate why fingerprinting is difficult to evade, and how resistance to fingerprinting would manifest itself. The value of script- and more generally content-blocking is slightly more difficult to estimate; as section 5.3 will show, this is a feasible approach, but an inconvenient and imperfect one. Section 5.4 will therefore investigate the value of *selective* content blocking. As Nikiforakis et al. only provide a perfunctory review of countermeasures other than user-agent spoofing, the later parts of this section will deviate somewhat from their paper. The third and fourth groups (according to Eckersley) will not be investigated further, as switching devices to visit websites suspected of fingerprinting is hardly practical.

5.2 – User-agent spoofing

When attempting to disguise one's identity, the logical starting point is the user-agent, which can be obtained both as HTTP header and via JavaScript as property of the `navigator` object. It contains a great amount of information about both the browser and its environment. Accordingly, Nikiforakis et al. tested 11 user-agent-spoofing browser extensions (8 for Firefox, 3 for Chrome) for the fifth chap-

ter of the *Cookieless Monster* paper. Extensions are distinct from plugins, and cannot be directly discovered via JavaScript enumeration, only their side-effects. (cf. Nikiforakis et al. 2013 p. 11)

The results of the investigation were less than encouraging. All of the (both popular and highly-rated) extensions tested by the authors failed to disguise the browser's identity sufficiently well. The reasons ranged from *incomplete coverage* (of the `navigator` object, with the `navigator.userAgent` property being changed but others remaining as before) to reporting *impossible configurations* (leaving the `screen` object unaltered and claiming mobile devices with impossible screen sizes); in some cases, the user-agent was altered in one place but not the other (e. g. the HTTP header was changed, but the `navigator.userAgent` property was not).

Furthermore, it is questionable if a browser extension even *can* achieve full coverage. For example, these extensions do not (and cannot) change the "special behaviors" discussed in section 4, but they would have to do this in order to present a manufactured identity that stands up to close investigation; vendor-specific features present a similar problem. (cf. Nikiforakis et al. 2013 p. 12) Again, even if an extension (e. g. TorButton) was to be extended to protect against the methods used by Nikiforakis et al. (e. g. altering the `screen` object as needed; cf. Nikiforakis et al. p. 6ff.), other methods remain against which it is difficult for a "mere" browser extension to defend without great amounts of inconvenience. An example for this would be the HTML5 `<canvas>`-tag based hardware fingerprinting described by Mowery/Shacham; while it would be possible to simply reject the `<canvas>` elements or constantly request permission from the user, this would represent a great loss in usability (cf. Mowery/Schacham 2012 p. 10).

Furthermore, due to the basic nature of fingerprinting, incomplete coverage can in fact be *worse than nothing* (creating an "iatrogenic problem"; cf. Nikiforakis et al. 2013 p. 12f.) If the goal is to achieve a sufficiently "non-unique" fingerprint, one should strive to blend into the masses. Using a "naive" user-agent-spoofing extension can have the opposite of the intended effect – *increasing* the fingerprintable surface by at least one feature, namely the fact that one is using a naive user-agent-spoofing extension. In fact, it may be possible to model "discrepancy patterns" and thus gain information about the *specific* sort of naive spoofing extension used, which would further worsen the situation. (cf. Nikiforakis et al. 2013 p. 12) Eckersley reached a similar conclusion in 2010, finding that the fingerprints of seven users who used a special "privacy-enhancing" browser were in fact *unique* (cf. Eckersley 2010 p. 14).

To summarize, current user-agent spoofing extensions are neither *robust* nor *complete* in their coverage. This directly contradicts Eckersley's findings that machines using TorButton were "relatively resistant" to fingerprinting (cf. Eckersley 2010, p. 16), indicating that the browser extension is (currently) insufficient to protect against more sophisticated methods of fingerprinting.

Therefore, it may be worthwhile to consider a different approach.

5.3 –Content blocking

As section 2 (and Table 1) have shown, many fingerprinting methods rely on Flash and/or JavaScript; without these, fingerprinting has to rely on HTTP headers and the presented IP address. Therefore, it might be worthy of investigation whether *content blocking* is capable of reducing the amount of information conveyed.

Globally disabling this content can be done fairly easily on most browsers, either through browser settings or with an extension or plug-in. (cf. Nikiforakis et al. 2013 p. 13) For Mozilla Firefox, for instance, the popular extension NoScript is available which blocks scripts and Flash content until explicitly allowed by the user (cf. NoScript website). The value of this blocking can easily be seen with the *Panopticlick* project; with NoScript active, it cannot obtain information pertaining to browser plugins, time zone settings, screen details or fonts installed. As anecdotal evidence, for the browser used by the author of this text, disabling JavaScript (via NoScript) decreased the amount of information conveyed from 20.58 to 13.9 bits, leaving the browser as "one in 15,297". No further privacy-protecting measures were taken, and it can safely be assumed that users of a less unusual user agent would profit more strongly.

This result may seem counter-intuitive at first, given the findings in 5.1 – after all, blocking JavaScript and Flash is *in itself* a fingerprintable feature, and on its own cannot make fingerprinting impossible, as the Panopticlick project demonstrates. However, it should be noted that many fingerprinting methods rely on JavaScript or Flash to a great extent, and much information cannot (easily) be obtained another way, which "cancels out" the additional feature.

However, this approach is not without limitations. It is not always feasible to simply block content like this, as it may be required for certain websites to function (e. g. if they rely on AJAX or use a Flash-based menu for navigation), and it is theoretically possible to integrate the fingerprinting code with these "first-party" elements. This requires a greater amount of coordination between first and third party, however. (cf. Nikiforakis et al. 2013 p. 13)

In the case of *tracking* (as opposed to *fingerprinting*), certain more "targeted" countermeasures exist already. The most successful approach so far have been manually-curated blacklists; browser expansions such as Ghostery are capable of blocking known trackers (cf. Ghostery website). There are several promising research directions beyond this, such as the automated learning of such blocking lists. Bau et al., for instance, presented such an experimental setup in early 2013 that resulted in 98% precision at a false-positive rate of 1% (cf. Bau et al. 2013 p. 1-5).

It may be possible to extend the machine learning approach to the domain of fingerprinting, even in cases where the fingerprinting code is very tightly integrated with "desirable" functionality. In a 2010 paper, Cova et al. presented a system for the recognition of malicious JavaScript based on dynamic script analysis and machine learning (cf. Cova et al. 2010 p. 1ff.). If appropriate features for the recognition fingerprinting code could be found (such as repeatedly querying for the same "type" of information such as user-agent in different ways), this approach could also be applied to the detection of fingerprinting functionality in scripts, and it may be possible to remove the offending sections from a script before executing it. However, at this point, this is pure speculation on the author's part.

6 –Discussion and conclusion

In their paper, Nikiforakis et al. investigated three commercial implementations of fingerprinting, created for different purposes but ultimately relying on similar methods.

They compared the implementations with one another and with Panopticlick, creating a broad taxonomy of fingerprintable features and showing the widespread use of Flash and JavaScript for (sometimes intrusive) fingerprinting.

In addition to the taxonomy, they investigated the actual use of fingerprinting scripts on the web, reaching the conclusion that adoption had started.

They further illustrated the usefulness of JavaScript for this purpose by drafting an experimental fingerprinting setup based on the special properties of the `navigator` and `screen` objects.

Finally, they analyzed popular user-agent spoofing extensions and showed that, for various reasons, they are ultimately incapable of disguising a browser's true identity and may thus end up making a browser *more* identifiable. Content blocking (specifically blocking of Flash object and JavaScript, universally or selectively) remains a somewhat effective stopgap solution, but more research in this direction is definitely necessary.

Fingerprinting is surprisingly difficult to evade. Thus, Nikiforakis et al. conclude, user privacy is currently "on the losing side", and the author of this text would be forced to agree.

References

(Bau et al. 2013): Bau , J., Mayer, J., Paskov, H., Mitchell, J.: A Promising Direction for Web Tracking Countermeasures, Stanford 2013

(BlueCava website): http://www.bluecava.com – No date, no author, accessed 2013-07-14

(Cova et al. 2010): Marco Cova, Christopher Kruegel, and Giovanni Vigna. 2010. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In Proceedings of the 19th international conference on World wide web (WWW '10). ACM, New York, NY, USA, 281-290.

(Eckersley 2010): Eckersley, P.: How Unique Is Your Web Browser?, in: Proceedings of the 10th international conference on Privacy enhancing technologies (PETS'10), 2010, p. 1-18

(Eckersley 2010b): Eckersley, P.: Browser Versions Carry 10.5 Bits of Identifying Information on Average, https://www.eff.org/deeplinks/2010/01/tracking-by-user-agent, accessed 2013-07-14

(Iovation website): https://www.iovation.com/ - No date, no author, accessed 2013-07-14

(Ghostery website): https://www.ghostery.com/ - No date, no author, accessed 2013-07-14

(Mayer 2009): Mayer, J.: “Any person... a pamphleteer" - Internet Anonymity in the Age of Web 2.0, Princeton 2009

(Mowery/Schacham 2012): Mowery, K., Schacham, H.: Pixel Perfect: Fingerprinting Canvas in HTML5, in: Proceedings of W2SP 2012. IEEE Computer Society, May 2012

(NoScript website): https://addons.mozilla.org/de/firefox/addon/noscript/ - No date, no author, accessed 2013-07-14

(Nikkiforakis et al. 2013): Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., Vigna, G.: Cookieless Monster: Exploring the Ecosystem of Web-based Device Fingerprinting, in: 2013 IEEE Symposium on Security and Privacy, 2013, p. 541-555

(ThreatMetrix 2013): No author (published by ThreatMetrix): Cybercrime Battle Basics - Online Account, Transaction and Device Protection: ThreatMetrix™ Cybercrime Defender Platform, 2013, http://info.threatmetrix.com/rs/threatmetrix/images/Cybercrime_Battle_Basics_Whitepaper.pdf, accessed 2013-07-14

(Veysset et al. 2002): Veysset, F., Courtay, O., Heen, O.: New Tool And Technique For Remote Operating System Fingerprinting, 2002