Research Internship: Towards a fingerprint surface

Rik Dolfing
Supervisor: Hugo Jonker

March 2019

Abstract

When fingerprinting is discussed, a lot of different terminology is being used. Studies use different terms, are not always clear about the studied features and propose methods which lack a fundamental approach. There is an understanding about the concept of fingerprinting, but it lacks a fundamental way to reason about it. We talk about a fingerprint surface and propose a taxonomy to use when discussing this topic. We applied our taxonomy to important studies over multiple years and reason about counter measures. We argue that the fingerprint surface is a fundamentally hard to problem, as it is hard to be complete in every category. We identified the one category that can be complete and we introduce the Prop-test to return the entire fingerprint surface for this category.

1 Introduction

Online tracking of individuals is becoming more popular than ever. This is because tracking of individuals has shown to be very lucrative e.g to find errors, count page views or maybe most important: targeted advertising. Tracking itself is not necessarily a bad thing when used for legitimate purposes such as fraud prevention, bot detection or improvement of security. However, for other purposes such as targeted advertising, tracking can be considered a destructive and privacy infringing method. A standard and widely used method is the use of cookies to create state-fullness across the web. Cookies can be used to store data on a visitors system and this data can later be read again by that domain to (re-)identify the visitor. As a visitor, you can clear your cookies, change your preferences about what is being saved (to a certain degree) and can even disable them. As cookies have their own regulation as of 2009[1], novel ways are explored to track users online. One of these methods is known as fingerprinting.

Fingerprinting itself is not exclusive to tracking, but it has become an important source of information for the ability to uniquely identify web visitors. Used in the context of the World Wide Web, we will refer to it as **browser fingerprinting**. Browser fingerprinting makes use of the properties or characteristics someone has when visiting a site. These properties can all be found somewhere in the TCP/IP OSI model. Browsers appear in the top layer: the Application layer. This means that data used for browser fingerprinting is gathered here as well and data in lower layers can not always be used. Javascript is build upon the application layer and can gather various features of website visitors. As visitors use different browsers, different versions, different hardware and different software, these properties can be fingerprinted to identify a visitor. This can be seen as intrusive behaviour as visitors might not be aware they are actually being tracked. Maybe even more important, fingerprinting is not something that can be easily turned off via the 'settings' in a browser or system.

[1]amended e-Privacy Directive 2002/58/EC

Most developed fingerprint methods lack a fundamental view for determining an actual fingerprint, as most methods seem to depend on 'randomly' chosen features of a visitor. These features are just a few when compared to the entire surface of which a fingerprint can be extracted, the so-called **Fingerprint Surface**. The fingerprint surface is the surface on which fingerprint techniques can work, to extract a range of features used to identify a user. Nikiforakis et al. [NKJ+13] were the first make make a mention about a **fingerprintable surface** and introduced fingerprint categories such as *Browser customisations, User configuration, Operating System and Applications* and *Hardware and Network*. They base their taxonomy on fingerprintable features which are currently being used by fingerprinters. They view it as a layered system, where on multiple layers information is extracted. It can be the case in their system that terms appear in multiple categories, which might cause confusion. Torres et al. [TJM15] were to first to actually define the term **Fingerprint Surface** and mention that it is almost impossible to capture the entire surface from which a fingerprint can be extracted. The studies limit their definition to currently found methods in the wild, meaning both Nikiforakis et al. and Torres et al. make use of a 'practical approximation' of the fingerprint surface. The practical approximation consist in both cases of an analysis of the attack surface, the surface used by commercial fingerprinters, to identify users. Neither consider the remaining possible features which could be extracted in the future. This means they limit themselves by not giving a fundamental view for determining the actual fingerprint surface. This makes them not directly comparable because of the the differences in studied features and usage, resulting in a practical approximation in both cases.

Furthermore, there appears to be a problem w.r.t. the classification of the fingerprint surface, as not all papers and websites appear to use the same definitions for terms related to fingerprinting. There seems to be a conceptual understanding of fingerprinting and the fingerprint surface, but it is unclear if studies mean the same if multiple terms are compared. Wikipedia makes a distinction between *passive* and *active* fingerprinting[2] and classifies passive as 'not obvious' querying and active as 'invasive'[3] querying. Nikiforakis [NJL15] talks about the *explicit* fingerprint, but does not give a clear definition. Multiple other studies mention *probing*[4] [FE15, LRB15b, BKST16] but do not give a definition what they see as probing. Additionally the same properties can be probed or be gathered in multiple ways [MM12, AJN+13], but do not state the differences. Another confusion is the difference between a *plugin* and *extension*, as this not always clear, as shown by Starov et al. [SN17], which looked into papers and saw the term being used interchangeable while it is not. A plugin is fingerprintable via the browser, while an extension is not. With all these definitions and different uses of them, it is unclear how much these studies and defences actually cover from a fingerprint surface perspective.

Contributions. In this study we will look into a fundamental way to reason about and describe the fingerprint surface. The main contributions of this work are:

- A new taxonomy based on a fundamental view, for reasoning about the fingerprint surface.
- A systemization of knowledge of multiple studies, which extracted a fingerprint or proposed countermeasures to existing fingerprinting techniques, based on this taxonomy.
- A proof of concept framework which extracts the fundamental browser properties fingerprint.
- A countermeasure section to explain the surface which countermeasures have to cover.

Limitations for our study are:

- Focused on browser fingerprinting, all the values should be retrievable in an Internet browser, on the application layer. Lower layers such as TCP/IP fingerprinting is therefore out of scope.

[2] `https://en.wikipedia.org/wiki/Device_fingerprint`
[3] Actively trying to gather information on a system
[4] Or similar term

- All findings from the Prop-test depend on Javascript. Moreover, Prop-test is currently limited to Google Chrome.

2 Background

Fingerprinting is a way to (uniquely) identify an individual being via properties or behavioural settings of that individual. In the case of browser fingerprinting, a site or domain tries to identify who is visiting or re-visiting by using unique 'features' of that visitor. This visitor can be human, but also be a bot, making fingerprinting an effective method to make a distinction between these two. As bots act differently on a website w.r.t static properties, mouse-movement and typing, fingerprinting can be used to analyse this kind of behaviour and identify a visitor accordingly. Properties of a visitor are harder to use in this case, as all bots can have the same settings, but is a good way to differentiate between humans, as they install different extensions, use different browsers and run a different hardware setup. All this differences and possible combinations of it, makes it possible for online trackers to uniquely identify users on the Internet. It is important to make the distinction between fingerprinting and profiling as the latter shows much less privacy implications. With fingerprinting the goal is unique identification, unlike profiling were the goal is to identify which 'group' is visiting e.g. age between 18-25, male or female or income $>$ 35k a year. These all say something about the visitor but do not make him uniquely identifiable. It should however be noted that all the combinations of these profiling groups, could in turn lead to an unique identification.

There are many ways to fingerprint visitors of a website. Most of the (browser) fingerprinters work client-side, but there are examples of server-side fingerprinters as well. Examples of server-side fingerprinting are the amount of requests, the contents of these request or the time between the requests. This is mostly implemented as a defense feature, as it can be used to detect bots, but will not be able to uniquely identify a visitor, as there is not that much to identify on. Hence, the client-side fingerprint surface is much bigger, as there are a lot of features to choose from which make someone identifiable. Below some examples on how this can be achieved.

2.1 Known fingerprinting techniques

Eckersley [Eck10] was one of the first to look into browser fingerprintable properties of a user using the site Panopticlick to fingerprint its visitors. A summary of some of these fingerprintable features are given below, including well-known other techniques:

- **Browser properties**: This method involves the fingerprinting of browser properties a user e.g. user-agent, HTTP-headers and content type. These methods take a combination of certain selected properties of the visitors browser and uses those to get extract a fingerprint.

- **Browser behaviour**: Depending on the installed versions of the JavaScript engine (or other different engines), the speed of these versions is different and might return different type of errors. When using this kind of fingerprinting, routines can be called to test the speed of the engine to determine the version and/or an error can created on purpose to see what the response is. As both are dependent on the installed version of Javascript in the browser with a lot of variety, these behavioural characteristics can be used in the identification of users.

- **Canvas API**: Canvas fingerprinting uses the power of the HTML5 canvas element to extract a fingerprint from this object. By rendering a piece of text with specified font size and background color into the canvas, extracting the pixel data and then take a hash of that data, a fingerprint can be obtained. Due to the different rendering methods from graphic cards and browser drivers this fingerprint has high entropy and is very usable as a fingerprint tool in a browser.

- **System behaviour**: This methods are used to extract the MAC address or serial numbers from a machine's hardware. This method is hard to utilize via a browser as these options are not directly available, but could be utilized via plugins. Most browser these days are protected against this type of fingerprinting. Other ways could be the usage of sounds or battery to determine unique identifiable features.

- **Battery**: The battery of a laptop or mobile device can be fingerprinted, using API calls to gather information about the battery state. This values decreases slowly and at some point increases to its maximum. Using this information, users can be followed around the web, by using their battery information. Unique identification this way is difficult, but it can be a good aid in combination with other features.

- **Extensions and plugins**: Flash, Silverlight, ActiveX are all examples of plugins which are used in fingerprinters. All of these have to be installed by the user to use them inside a browser. Especially Flash is used for fingerprinting purposes like determination of the available fonts but also version enumerations of plugins can be used, as there are no automatic updates with a lot of versions, making it a perfect fingerprintable feature.

- **User behaviour**: Users behave differently on the Internet, not only in the aspects of what sites they visit, but also w.r.t their mouse movements, mouse clicks or typing rhythm. Using Javascript, the acceleration, curvature distance and angle of curvature can be used to fingerprint a visitor. These features alone are not enough, but can aid in combination with other features like the one's named above.

All of the above are examples of how website owners will try to fingerprint its visitors. There are many more methods out there and new methods are constantly being introduced to the world and deployed in the wild. It is more or less an arms-race, as each time a new method arises, a new countermeasure follows shortly after. One of the biggest problems w.r.t countermeasures is that, paradoxically, it might make its user more unique. As for one of the biggest problems for attackers is that the manufacturers are starting to implement restrictions on functionality of certain features. These functionalities were used to generate a unique fingerprint in the past, an example is Apple, which leaves out detailed information of certain software versions. Instead of reporting version 2.14.2125, it will now report version 2.[5] This type of countermeasure will try to make its user less unique and therefore will blend it with the rest of the crowd. Other countermeasures which have been proposed such as supplying 'wrong' information to a website by randomizing the user-agent, or always display the same image on the canvas element.

3 Taxonomy

As can be seen in the background, there are a lot of fingerprinting examples. When studying the current literature, it is hard to reason about the entire fingerprint surface, because most papers focus on only a small part e.g. building a defence against a certain type of attack, instead of building a defence against the entire surface. Below we introduce a taxonomy to reason about the fingerprint surface, using a novel approach which we believe to be helpful when developing counter measures or reason about privacy implication. While Nikiforakis et al. [NKJ+13] and Torres et al. [TJM15] reasoned about a fingerprint surface from an attack vector, a more practical approach while we will take a more fundamental approach. This fundamental approach looks at entire surfaces for browser fingerprinters to work on, unlike the practical approach, which looks at what is done by known techniques and limits itself to those features. Below (See *Table 1*) we introduce a new system to reason about the reach of a fingerprint surface. This categorisation is based on techniques found in studies from the Systemization of Knowledge. By studying those papers and examining the 'evolution' of fingerprinters, we came up this taxonomy. As fingerprinters started to operate outside of a browser window, new surfaces for fingerprinting were discovered. All currently known techniques can be placed into one of the six categories below and will only match one category each.

[5] `https://www.apple.com/newsroom/2018/06/apple-introduces-macos-mojave/`

To understand why this taxonomy has been chosen, first some clarification of certain terms and how we use them in our taxonomy[6]:

- **Probing:** Actively requesting values, which are known to the requester beforehand. The requester knows what he is looking for and sees if they are available. Values which are returned directly, do not fall under this definition, one does not need to probe each separate value, to see if it exists.
- **Browser Extension:** Browser extensions are meant to extend or modify HTML. Extensions are not directly detectable, but the observed changes in the DOM might indicate their existence.
- **Browser Plugin:** Plugins allow browsers to parse and display content that is not traditional HTML. Plugins are available without probing in a browser, using simple Javascript.

With this definitions and the terms shown in *Table 1*, below a list with explanation for all of the categories, followed by a section about some ambiguous examples:

- **System properties**: All properties specific to a system, independent of users installed additions or installed browser. An example is a users screen resolution, this resolution is based on the monitor attached to the system, but switching to a different monitor will give the system a different screen resolution. Also values which are based on the system, but available via the browser fall under this category.
- **System behaviour**: All behaviour that is bound to a specific system and can be based on hardware or software behaviour. Font probing is the best known example. It probes a list of known fonts and tests the systems responses and return order of the fonts. Another example is error probing, using different error test cases and see how the system responds to certain commands. This behaviour is system specific and does not depend on the browser.
- **Browser properties**: All properties directly relying on a browser and its implementation on a software level. All browsers send variables with their requests to make the server know who is connecting and how to handle the connection. All the values that are specific for a browser are placed inside this category.
- **Browser behaviour**: All the behaviour that can directly be observed in a web browser regarding rendering and errors. Examples are Canvas fingerprinting and Javascript error observations. By drawing inside the canvas element, the browser uses certain techniques which are mainly dependent of the browser itself, which make the drawing appear differently, or shows different errors.
- **User properties**: All properties which specifically lead to a single user and which acts as personalization of a user. Installed plugins, visited site cookies, detected extensions and installed custom fonts are all examples. It is somewhat of a choice that a user has this installed or available on his/her PC, making it something which a user processes, a property.
- **User behaviour**: This is based on a how a user behaves by studying his typing pattern, mouse movements or mouse clicks. A comparison could be the human voice, humans can say the same sentence, but are identifiable how they say it regarding sound, pronunciation and other recognizable person specific traits.

Using above system, it is possible to reason about the fingerprint surface which is being covered and where the focus is on. It gives a formal way to reason about what type of fingerprinting a countermeasure actually defends against and what needs to be done in the future to reduce the fingerprint surface in multiple categories. It should be noted that most studies and browser fingerprinters do not limit themselves to only a single category e.g. Eckersley [Eck10] is using browser properties, browser behaviour, system behaviour and user properties.

[6]For Browser Extension and Browser Plugin we took the definition from [SN17]

	Properties	**Behaviour**
System	Screen Resolution	Font probing
Browser	User-agent, HTTP-accept	Canvas fingerprinting
User	Cookies, plugins, custom fonts	Mouse movements, typing rhythm

Table 1: Taxonomy of a fingerprint surface with examples for each category.

3.1 Priority and Ambiguous cases

There are arguments to be made for certain features to be in multiple categories and some might even change depending on technological changes.

Font probing

Fonts need to be probed now to get the desired answer, but maybe later it becomes available via a simple Javascript call. Another possibility, also for fonts, is that the user could install custom fonts onto the system, which theoretically could be probed for.[7] This would put fonts into two separate categories: user properties but also system behaviour. For fonts this can be solved in the following way: System fonts for system behaviour and custom fonts for user properties. While custom fonts need to be probed as well, we feel that a user property or user behaviour always has *priority* in our taxonomy. If it is possible that a property is based on a users choice and a personalization for a specific user, we will assign it as such. In this way we make sure that the customisation of a user is reflected properly in the fingerprint surface, as this is an important factor which can be fingerprinted. Examples include installing extensions which block advertisements, but if a user installs such a extension, he might become more unique.

Canvas fingerprinting

A difficult case is the one of Canvas Fingerprinting. It is dependent on both the systems behaviour and browsers behaviour. This is because of the video adapter, the installed (browser)drivers and the browser which renders this. We feel canvas fingerprinting is something which belongs to a browser. While true that is based on system behaviour in some way, the reason is it also rendered different in multiple browsers on the same hardware makes us believe it is more of a browser property, than a system property. To strengthen this argument, canvas fingerprinting can exclusively be used in a browser, by using the `canvas` element. Canvas fingerprinting could potentially be classified as system behaviour in the future, if the behaviour becomes solely related to the video adapter.

4 Systemization of knowledge

We will look into what has happened from 2010 when Eckersley released his study, up until now and what can be observed in the meantime. First some (highlighted) papers will be discussed in chronological order. We apply our taxonomy on each paper and give an overview of the fingerprint surface (if possible) for each paper.[8] After that there will be a short overview about targeted countermeasures and what should be done on a fundamental level to reduce the fingerprint surface. Lastly, what needs to be done for entire fingerprint surface coverage. In all these cases, the focus will be on browser fingerprinting, with some sidetracks to show a different environment or a different take on fingerprinting to elaborate on some parts of the fingerprint surface.

[7] We didn't find evidence for this happening in the wild.

[8] Not each paper is explicitly stating the *exact* features, we state the features that are being named in the paper. Some papers only name 'Screen' while others report screen- width, height and depth separately. Sometimes the language is explicitly being named as browser language, while others specify just 'language'

[Eck10]: First major browser fingerprinting study

Browser fingerprinting started to gain traction around 2010, when Eckersley presented his paper: "How Unique Is Your Web Browser?" [Eck10]. He showed that 94.2% of the Panopticlick visitors were uniquely identifiable. Not only did he find this, he also showed that even though fingerprints change rapidly, he could match the 'old' and new fingerprint with $\geq 99\%$ certainty. Eckersley's fingerprint surface is primarily focused on browser properties, but uses font and plugin enumeration, which respectively belong in the system behaviour and user properties category. He also looks into the timezone and screen resolution, which belong in the system properties category. He mentioned the most entropy comes from fonts and plugins, as they can take on the most values. While fingerprinting in general is nothing new, as discussed by Eckersley in his paper, browser fingerprinting really took of from this moment on.

FP surface:

- Browser properties:	User-agent, HTTP-ACCEPT, Cookies enabled, MIME types
- System properties:	Screen resolution, timezone
- System behaviour:	System fonts
- User properties:	Browser plugin, plugin version

[BFGI11]: Cross browser tracking study

In 2011, Boda et al. [BFGI11] proposed an improvement on Eckersley's study, a novel method based solely on Javascript and which could work cross browser. Important for us to notice is that they indirectly introduced a differentiation between font sets. They differentiate between universal and basic fontlists. Universal fonts work good independent of browser and were introduced in the paper because they *always* work on any browser. This can however also be turned around, *if* you have a non-universal font, you become more unique.[9] In the end, they came to the same conclusions as Eckersley, without using plugins fingerprints.

FP surface:

- Browser properties:	User-agent, HTTP-ACCEPT, Cookies enabled, MIME types,
- System properties:	Screen resolution, timezone
- System behaviour:	System fonts
- User properties:	**Non-universal fonts**

[MRH$^+$12]: Fast and lightweight JS engine identification

Most studies put their efforts into building Javascript code, as this has proven to be a reliable and more future proof way to generate a fingerprint, without too much loss of functionality. Flash still is present, but support stops in 2020.[10] Mulazzani et al. [MRH$^+$12] were not the first to try to identify the Javascript engine used, but presented a lightweight and fast solution to this problem. They used a decision tree based on the failed test cases from *Test262*.[11] Added benefit was that they were able to detect spoofed user-agents, because it is almost impossible to spoof an Javascript engine version which is linked to a certain browser. Their fingerprint surface is solely present in browser behaviour, they only test the behaviour of the Javascript engine.

FP surface:

- Browser behaviour:	**Test262**

[MS12]: First Canvas Fingerprinter

Mowery and Shacham [MS12] first talked about the close correlation of hardware and browsers. Their technique, now more commonly known as canvas fingerprinting, revolved around the rendering of text and WebGL scenes to the `canvas` element and examine the pixels produced. Because there is so much entropy and it is orthogonal to other fingerprints, it is a good extension to other

[9] Not exactly concluded in the paper, but a takeaway we learned from reading the paper

[10] https://theblog.adobe.com/adobe-flash-update/

[11] https://github.com/tc39/test262

known fingerprinting techniques. Mowery and Shacham's fingerprint surface falls in the browser behaviour category, as they are solely relying on a browser behaviour from a canvas element.
FP surface:

- Browser behaviour:	**Canvas Fingerprinting**

[MM12]: Extensive survey of tracking and policies.

Mayer and Mitchell [MM12] present a survey on the current state of technologies. They showed via the Fourthparty[12] framework, what techniques are used in the wild and what researchers can focus on in their future research. They looked into third-party tracking policies, regulation and self-regulation in the US and in Europe, analysed business models and finally tracking technologies such as cookies and fingerprinting. They talk about the user choice mechanisms such as Do Not Track and usage of it. With their framework make us of properties only, as they are not probing for variables.
FP surface:

- Browser properties:	HTTP Traffic, DOM: `navigator`, `window`, resource loads.
- System properties:	Screen
- User properties:	Cookies

[NKJ+13]: Examining three commercial fingerprinters and reveal questionable practices, first taxonomy for fingerprinting.

It became more widely know that fingerprinting techniques were being deployed in the wild, partly because cookie regulations started to kick in[13]. When in 2013 Nikiforakis et al. [NKJ+13] analysed the code of three popular browser-fingerprinters, also the questionable practices of fingerprinting were given more attention. While on one side Mowery et al. [MS12] showed the combat of fraud, the destructive side of track users became more popular. Nikiforakis shows the circumvention of proxies to instruct Flash to directly contact the server. They also report that defenses are lacking in the coverage, can create impossible configurations and might even aid current fingerprinters, as these countermeasure plugins might make one more unique. It can however be concluded that unlike Eckersley, fingerprinters more and more move towards a fingerprint surface starts to behaviour. While Eckersley only looked at fonts in this category, these commercial fingerprinters also explored: driver enumeration, math constants, Windows Registery and AJAX implementation as system behavioural properties.
FP surface:

- Browser properties:	User-agent, HTTP-ACCEPT, Cookies enabled, MIME types, **Do-Not-Track, Browser language**
- Browser behaviour:	**Math Constants, AJAX implementations, Flash manufacturer**
- System properties:	Screen resolution, timezone, **Proxy detection, Windows Registery, IP-address, MSIE Product Key, OS+kernel version**
- System behaviour:	System fonts, **driver enumeration, TCP/IP parameters**
- User properties:	Browser plugin, plugin version

[AJN+13]: Detection and analysis framework for web-based fingerprinters

Acar et al. [AJN+13] created a framework called *FPdetective*, a framework for detection and analysis of web-based fingerprinters. Their main focus in on font detection and found that Bluecava is the most used fingerprinter. They also showed that most of the fingerprinters relied mostly on Javascript and saw that the amount of Flash fingerprinters were significantly lower (400 vs 100). They report a complete disregard towards the Do-Not-Track header. As for the use of Tor and

[12] `http://www.fourthparty.info`
[13] `https://ec.europa.eu/justice/article-29/documentation/opinion-recommendation/files/2013/wp208_en.pdf`

Firegloves(a plug-in for Firefox to randomize values), both were not equipped well enough against font enumeration due to covering to few features or simply not working.
FP surface:

- Browser properties:	User-agent, appCodeName, product, productSub, vendor, vendorSub, onLine, appVersion, browser language, MIME type-objects(enabled plugin, description, suffixes, type), cookieEnabled, javaEnabled
- Browser behaviour:	CSSFontFace::getFontData, CSSFontSelector::GetFontData
- System properties:	Screen(horizontalDPI, verticalDPI, height, width, colordepth, pixeldepth, availLeft, availTop, availHeight, availwidth)
- User properties:	Browser plugins (name,filename,description,length),

[AEE$^+$14] : Advanced tracking mechanics in the wild

In 2014, Acar et al. [AEE$^+$14] looked into three advanced web tracking mechanics: canvas fingerprinting, evercookies and cookie syncing and show that canvas fingerprinting is gaining in popularity. Canvas fingerprinting can be found at 5% of the sites in the Alexa 100.000. Evercookies were found in the top 3000 sites and IndexedDB was found as a new attack vector. This probe-able attribute can be used to find evercookies. Cookie syncing (with the use of evercookies) showed a lucrative way to match multiple cookies to one user ID to gain more information on a visitor and was found 101 times(out of 3000 sites). For the fingerprint surface, canvas fingerprinting is most important which falls under browser behaviour. Cookies can be put under the user properties, as those change depending on the user his Internet visits.
FP surface:

- Browser behaviour:	Canvas fingerprinting, **IndexDB exist**
- User properties:	Cookie syncing, **IndexDB Evercookies**,

[ZDLZ14]: Fingerprinting sound

Zhou et al. [ZDLZ14] showed that is was possible to capture speaker sound of a device, via its own (or different) microphone. Creating sounds from the device, which are impossible to hear for humans, then capture and analysis this, a fingerprint can be extracted and a device hardware fingerprint can be created. While experimental, it showed promising results which could eventually be used in the wild. Their fingerprint surface is within the system behaviour category, as different hardware and software components use a different method to analyse sounds.
FP surface:

- System behaviour:	**Processing of sound**

[LRB15a]: Second major browser fingerprinting study

Starting 2015, Laperdrix et al. [LRB15a] created an updated version of Eckersley's Panopticlick, called Amiunique.org, implementing new advanced techniques into this fingerprinter e.g. canvas fingerprinting and WebGL renderer. They also tested this, as one of the firsts, on mobile devices and showed that even without fonts and plugins, mobile fingerprints can be very recognizable by a verbose representation of the user agents and emoji's. They showed >80% unique mobile fingerprints and once again confirmed Eckersley findings(>90% unique) for the browsers for desktops. They didn't introduce new features, but are the first to incoorporate some of these in a large scale study such as: Content Encoding, Concent language, Do-Not-Track, WEBGL Vendor, WebGL Renderer, List of HTTP-header.
FP surface:

- Browser properties:	User-agent, Cookies enabled, MIME types, Content Encoding, Content language, Do-Not-Track, WEBGL Vendor, WebGL Renderer, List of HTTP-header
- Browser behaviour:	Canvas fingerprinting
- System properties:	Screen resolution, timezone,
- System behaviour:	System fonts
- User properties:	Browser plugin, plugin version

[OACD15]: Fingerprinting the battery of a device

Another example of the correlation of hardware and software was shown by Olejnik et al. [OACD15], as they were able to differentiate users based on their battery. While this only works for devices which have a battery, it shows a possible fingerprint surface because of the high amount of entropy. Firefox accepted their bug report and now is less verbose in this regard, but this example shows just how this kind of techniques can be used to track users on the web. The fingerprint surface would fall under the system property category, as this value can simply be queried via Javascript.
FP surface:

- System behaviour:	**Battery**

[NJL15]: Random non-consistent fingerprint

PriVaricator by Nikiforakis et al. [NJL15] uses randomized variables focussing on fonts and plugins, randomizing return data from fonts offsets and about the amount of plugins. The defensive measures showed promising results for unlinkability, but only looked into features that are being used by major fingerprinters such as BlueCava and AddThis. They also state that randomization is a double edged sword: report random values which are not in line with expected values, could make one more vulnerable or could crash in certain cases. PriVaricator is focused on the findings in these commercial tools, randomizing certain trackable variables. They state that it how is easily detectable if sites test for unexpected randomness.
FP surface:

- Browser properties:	offsetHeight, offsetWidth, getBoudingClientRect
- User properties:	amount of plugins

[TJM15]: Random consistent fingerprint

More and more defences against fingerprint ability became available such as FPBlock by Torres et al. [TJM15]. They want to create unlinkability of a fingerprint over time while preserving a consistent fingerprint between multiple visits of the same site. Defensive measures showed promising results for unlinkability, but they only looked into features that are being used by major fingerprinters, like PriVaricator did. They also state that randomness can be a double edged sword: report random values which are not in line with expected values, could make one more vulnerable. FPBlock creates a consistent fingerprint, which makes it a direct improvement over PriVaricator, while guaranteeing the same privacy.
FP surface:

- Browser properties:	User-agent, Cookies enabled, MIME types, Content Encoding, Browser language, Content language, Do-Not-Track, Cookies enabled, Flash enabled, WebGL Renderer context, List of HTTP-header, Flash enabled, OpenDatabase,
- Browser behaviour:	Canvas fingerprinting, IndexDB
- System properties:	Screen resolution, timezone, OS&Kernel-version, CPU
- System behaviour:	System fonts, Battery
- User properties:	Browser plugin

[LRB15b]: First virtualization study fingerprint unlikablility

Blink is a tool created by by Laperdrix et al [LRB15b]. Blink uses Virtual Machines(VM) for a changing fingerprint overtime as these can be easily reconfigured, stating they don't have to trick websites, they just update their virtual environment. They used their results from a previous study, Amiunique [LRB15a], to generate fingerprints based on their findings of 'legit' fingerprints. Blink tends to use use a more fundamental approach by changing the entire configurations, using virtualization to set certain variables.
FP surface:

- Browser properties:	User-agent, HTTP-ACCEPT, Cookies enabled, MIME types, DOM-storage
- System properties:	Screen resolution, timezone, OS&Kernel-version
- System behaviour:	System fonts
- User properties:	Browser plugins, cookies

[FE15]: Extensive font difference study

Fifield and Egelman [FE15] looked into more extensive font identification. While most previous studies stopped at only checking for the availability of certain fonts [Eck10, BFGI11, NKJ+13, AJN+13, LRB15a] or rendering fonts into a canvas [MS12, NKJ+13, LRB15a], they decided to measure the onscreen size of font glyphs and tested almost the entire repertoire of Unicode. They then narrowed this down to the best identifiers, which tended to be around 40. For the fingerprint surface, their research would fall into the system behaviour category, as they only use system behaviour features to determine this kind of fingerprint.
FP surface:

- System behaviour:	System fonts

[SPK16]: Major recaptcha analysis

In 2016, Sivakorn et al. [SPK16] used deep learning to solve captchas. There has been a lot of work into captchas as they are a method to stop bots from doing automated activities, by presenting challenges to visitors, hard for bots and easy for humans. As they are severely hurting the browser experience for visitors, constant improvements are made. One of this improvements was a risk analysis, based on multiple fingerprintable properties of a user [Emb]. Sivakorn et al. focused their study on Google's reCAPTCHA[14], which is still the most used recaptcha service. They found that multiple factors weighted in when presented with a so called checkbox captcha. They found that the user-agent should always return a possible combination, and wrong environment variables, misinformation or misformatted would result in fallback captchas. Also cookies were an important source, and mouse behaviour was not taken into account by the risk analysis at all. Suggesting the current fingerprint surface for the reCAPTCHA system is browser properties, user properties and system properties. In the end they managed to solve fallback image captchas with deep learning, so even if the checkbox captchas would fail at some point, image captchas could automatically be solved.
FP surface:

- Browser properties:	User-agent, Cookies enabled, Webdriver type
- System properties:	Screen resolution
- User properties:	Browser plugin, user account
- User behaviour:	Mouse movement

[EN16]: Introduction of OpenWPM

Englehardt and Narayanan [EN16] did 15 different measurements, including stateful(cookies) and stateless(fingerprinting) techniques, by using OpenWPM (a tool introduced in their paper, which continues the work from FourthParty [MM12]). OpenWPM is a privacy preserving framework

[14] `https://developers.google.com/recaptcha/docs/versions#invisible`

which is an automated version of a full-fledged browser. They measured all the features named before by previous works such as canvas fingerprinting, battery fingerprinting and fonts [Eck10, BFGI11, MS12, OACD15, ZDLZ14, FE15] and with new inclusions of AudioContext(probe-able) and WebRTC-based fingerprinting. They also report the same findings about cookie syncing as Acar et al. [AEE$^+$14], but on a larger scale, as almost all third parties are involved in this kind of cookie activities. Looking into the fingerprint surface for this specific study, all would be inside system behaviour and in the user properties category. However because of the extensible framework, this is only true for this study.

FP surface:

- Browser properties:	HTTP Traffic, DOM: `navigator`, `window`, resource loads, WebRTC
- Browser behaviour:	Canvas fingerprinting, AudioContext enabled
- System properties:	Screen
- System behaviour:	Audio processing
- User properties:	Cookies

[BKST16]: Chromium browser investigation, using multiple hiding strategies.

Baumann et al. [BKST16] looked into the protection of the Chromium browser w.r.t Flash and Canvas fingerprinting protection. They also introduce two different strategies: Many browsers, one configuration and one browser, many configurations. They showed that it is possible to fool trackers without randomization or blocking and state limitations of current fingerprint blockers. The covered fingerprint surface is within the system behaviour category for canvas fingerprinting and for some Flash options like font probing, and system properties for the rest of the Flash properties.

FP surface:

- Browser properties:	App Version, Flash version, Browser language, MIME types, User-agent
- Browser behaviour:	Canvas fingerprinting,
- System properties:	Screen (Height, Width, color depth, pixel depth, resolution), Windows version
- System behaviour:	System fonts
- User properties:	Browser history, plugins

[SN17]: Xhound, a framework to detect extensions

Starov and Nikiforakis [SN17] looked into fingerprintability of browser extensions and discovery of them. Starov created a framework Xhound to create signature for the top 10.000 most popular Google Chrome extensions and use those to identify users based on their installed extentions. This is done by detecting changes in the DOM. They note the difference between a plugin and an extentions as follows: “Plugins allow browsers to parse and display content that is not traditional HTML ... browser extensions are meant to extend or modify HTML.” They gave countermeasures such as encapsulation and namespace pollution and argue that extension fingerprinting is more intrusive than other fingerprinting techniques. In one way because users take they extensions with them when switching to a different machine in Chrome and extensions have more entropy than canvas fingerprinting and this is already widely adopted.

FP surface:

- Browser Behaviour:	**Extension testing**

[SAS17]: ID based extension detection

Sjorsten et al. [SAS17] looked into discovery of browser extensions via static web manifest calls. They looked at unique id’s for each extension. They showed that there are already websites using this method as well. Their detection is purely based on a unique identifier making it a non-behavioural detector. Limitation is that not all extensions use this web manifest, making it incomplete as a extension detector. *Additionally*, in our taxonomy, this actually behavioural testing. As all the possible values for each extension have to be probed, using a predefined list of all possible extensions. The return value will indicate if the extension is available.

FP surface:

- System Behaviour:	**Web manifest probing**

[BBW17]: Mouse-movements based gender identification

Van Balen et al. [BBW17] studied the behavioural differences on the web between males and females using mouse movements. In 2001, even long before Eckersley, Mueller and Lockerd [ML01] already looked into using mouse movement to model a user interests and found that certain movements and behaviour suggests certain interests. This time, van Balen et al. used this kind of mouse movements patterns identify genders and found that they are able to identify around 75% correctly and showed the most important features for this : peak velocity, length of deceleration phase, accuracy, finger posture and reaction time. This study is not explicitly aimed at tracking users, but can directly be translated to a scenario were this is possible, or serve as a extra feature in uniquely identifying users. Their fingerprint surface is in the user behavioural category.

FP surface:

- User Behaviour:	Mouse-movement

[LBM17]: Applying randomness in advanced fingerprinting techniques

Laperdrix et al. [LBM17] present FPRandom, a new form of creating unlinkability of fingerprints by introducing different forms of randomness. They included defences against the newest techniques such as canvas fingerprinting, `AudioContext` fingerprinting and the order of certain properties of Javascript. They showed that introducing small amount of randomness indeed reduces the fingerprint stability. A catch is that FPRandom only works on advanced techniques and not on the basics covered by FPBlock [TJM15] and PriVericator [NJL15]. The fingerprint surface covered are all system behaviour.

FP surface:

- Browser behaviour:	Canvas fingerprinting, **Javascript objects ordering**
- System behaviour:	Audio processing

[GBLL18]: Third major fingerprinting study, with a twist

Most recent, in 2018, Gomez-Boix et al. [GBLL18] tried to recreate the findings of Eckersley once again. Surprisingly, they found only a 34% unique identification rate. They collected over 2 million samples for the top 15 French websites. They argue that all previous work is biased, as Amiunique [LRB15a] and Panopticlick [Eck10] are focused on privacy aware people. They also report that although they are less unique, only small changes need to be made to the fingerprint to make them unique. The tested features are the same as in the Amiunique [LRB15a] study. Hence the fingerprint surface is the same as well.

FP surface:

- Browser properties:	User-agent, Cookies enabled, MIME types, Content Encoding, Content language, Do-Not-Track, WebGL Vendor, WebGL Renderer, List of HTTP-header
- Browser behaviour:	Canvas fingerprinting
- System properties:	Screen resolution, timezone,
- System behaviour:	System fonts
- User properties:	Browser plugin, plugin version

[VRR18]: Short- and longterm fingerprinting framwork

Vastel et al. [VRR18] present FP-Tester, automated testing of fingerprinting resilience. Using FP-tester, countermeasures can be tested as it reports side effects as well as the impact of tracking.

Unlike XHound [SN17], which mainly looks at the DOM changes, FP-Tester looks for inconsistencies, overridden native Javascript and manipulated canvas pixels. Short-term changes can be observed in the extensions, while FP-Tester aids in long-term changes as well, as it compares fingerprints during the test process. It tests for the same features which are reported by Amiunique.

FP surface:

- Browser properties:	User-agent, Cookies enabled, MIME types, Content Encoding, Content language, Do-Not-Track, WEBGL Vendor, WebGL Renderer, List of HTTP-header
- Browser behaviour:	Canvas fingerprinting
- System properties:	Screen resolution, timezone,
- System behaviour:	System fonts
- User properties:	Browser plugin, plugin version

[FvGJ18]: Cookie policy and implementation analysis

Franken et al. [FvGJ18] showed that policies are bypassed easily and they introduce new techniques to bypass defences as well. Their introduced automatic framework, to find cross-site countermeasures and anti-tracking policies of 7 browsers and 46 browser extensions. More than 90% of those could be bypassed using a set of techniques described in their study e.g by using build-in PDF readers or simply flawed implementations. They concluded that browsers have inconsistent behaviour and are all prone to inconsistencies due to the freedom of implementation. As this is mainly an evaluation of cookie policies, the fingerprint surface is limited to those as well, meaning everything falls under the user property category.

FP surface:

- User properties:	Cookies

[Vlo18]: Detecting web-bots using fingerprinting techniques

Vlot [Vlo18] looked into ways to detect web-bots and scrapers. Vlot took a fingerprint approach, identifying what properties are unique to a web-bot and also created detection patterns based of that. He showed which type of bots are easily identifiable and in what setting. He found that 13% of of the websites used some kind of fingerprint method to identify if there were being scraped.

FP surface:

- Browser properties:	User-agent, MIME types, Content Encoding, Content language, WebGL Vendor, browser language, `navigator` attributes
- Browser behaviour:	Canvas fingerprinting
- System properties:	Screen (color depth, resolution), timezone,
- System behaviour:	System fonts
- User properties:	Browser plugin

[MDM$^+$18]: Client-side and server-side identification

Marques et al. [MDM$^+$18] compared two different tools, where one was mainly client-side and the other was server-side. The server-side tool solely relied on HTTP logs and was more useful for longer connections and creating correlations between multiple visits. The client-side tool was faster at identifying bots. As it is not disclosed which tool is being used, it is hard to determine the exact fingerprint surface for this. The main take away is that server-side fingerprinting can be used to detect bots in the long term, while this is hard for client side.

Table 3 shows an overview of all the fingerprint surfaces of the above discussed papers.

Paper	SysProp	SysBeha	BrowsProp	BrowBeha	UserProp	UserBeha
[Eck10]	▨	▨	🟥		▨	
[BFGI11]	▨	▨	🟥		▨	
[MRH+12]				🟥		
[MS12]				🟥		
[MM12]	▨		🟥		▨	
[NKJ+13]	🟥	▨	▨	▨	▨	
[AJN+13]	▨	▨	🟥		▨	
[AEE+14]		🟥			▨	
[ZDLZ14]		🟥				
[LRB15a]	▨	▨	🟥	▨	▨	
[TJM15]	▨	▨	🟥	▨	▨	
[NJL15]		🟥			▨	
[LRB15b]	▨	▨	🟥	▨	▨	
[FE15]		🟥				
[OACD15]		🟥				
[SPK16]	▨		🟥		▨	▨
[EN16]	▨	▨	🟥	▨	▨	
[BKST16]	▨	▨	🟥	▨	▨	
[SN17]				🟥		
[SAS17]		🟥				
[BBW17]						🟥
[LBM17]		▨		🟥		
[GBLL18]	▨	▨	🟥	▨	▨	
[VRR18]	▨	▨	🟥	▨	▨	
[FvGJ18]					🟥	
[Vlo18]	▨	▨	🟥	▨	▨	

Table 2: Overview of the found fingerprint surface per paper,
red = primary focus, gray = covered

4.1 Countermeasures

4.1.1 Targeted defences

Looking at all the targeted defences or frameworks reviewed by this study: FPDetective [AJN+13] FP-block [TJM15], Privaricator [NJL15], Blink [LRB15b], OpenWPM [EN16], FPRandom [LBM17] and FP-tester [VRR18], all focus on targeted defences, only to defend against known properties of web fingerprinters such as BlueCava. This was also studied by Luangmaneerote et al. [LZC16], where they compared multiple of these countermeasures and showed mixed results about effectiveness, usability and inconsistencies. This does not necessarily mean that these defences did not work in the past, but commercial fingerprinters have evolved, finding new ways to extract a fingerprint or use other methods to determine if there is some kind of defensive mechanism active. This can be done by comparing the `navigator` object, user-agent string and HTTP header. This shows that this is an arms race between third party fingerprinters and defences from the academic world. To illustrate this even more, Nithyanand et al. [NKJ+16] found that already 50% of the origional adblockers, can circumvent anti-adblockers.

4.1.2 Fundamental defences

There are methods in which fingerprinters become less effective. Below is a list of such suggested countermeasures on a more fundamental level by papers who did not introduce a framework or defensive extension themselves:

- **Block JS/Flash:** This is the most straight forward solution to simply block Javascript and Flash. While not ideal for a users browser experience, this might be the best solution, as all fingerprinters rely on some form of Javascript or Flash.
- **Use Tor:** Use of the Torbutton extension or the Tor browser. From the beginning, Tor is focused around anonymity and privacy. They incorporate targeted defences from the academic field and also are proactively implementing defences against canvas fingerprinting and use permission management.
- **Ask permission:** One of the most obvious solutions, however severely negatively impacting the browser experience: ask permission if you need attributes to fingerprint and let the user decide upon this. Tor and Firefox.
- **Native browser defence:** This would basically mean implementing the targeting defences from above, inside the the core of the browser. This way everyone has the same plugins, uses the same defences and removes the paradox of becoming more identifiable via installed plugins.
- **Standardization:** Because there are so much different JavaScript engines, Flash versions, popular plugins such as AdBlocker, there is an argument to be made to make this behave the same across multiple browsers. While this is a viable solution, it is also really hard to coordinate between vendors and hard to enforce. It will also be prone to inconsistencies as is already the case multiple defensive plugins.

5 The 'entire' surface, a fundamentally incomplete problem

It can concluded that fundamental countermeasures are hard to enforce or would severely ruin the user experience. Blocking of Flash and Javascript are not really solutions, but more a necessary evil. Switching to Tor is inconvenient for 'normal' Internet users, however, more standard browsers start to implement Tor specific features like letterboxing[15] and permission management for certain operations like canvas access. Lastly, standardization and native browser defences are hard to achieve between multiple commercial parties, as they have their own vision and implementation preferences.

Currently, there are two scenarios which have the possibility to solve the 'entire' fingerprint surface problem, but both bring their own difficulties:

1. Every browser should look the same, so fingerprinting becomes ineffective.
2. Every browser should protect against fingerprinters using defensive mechanism.

Currently, we are mostly doing *2*, defending against imminent threats. But as mentioned in a paper [BKST16], *1* might be were we want to go. For now, users will have to rely on targeted defences, but this seems unsustainable for both attackers and defenders, as the arms race keeps continuing. If everyone is the same in both properties and behaviour, fingerprinting makes no sense, as there will not be uniquely identifiable features left. The problem is that it will introduce significant overheat in how we experience the Internet via a browser by introducing virtual machines, special input methods to hide your mouse and typing and latency. A trade-off might offer a solution, but with the current developments in the field, it is hard to give an answer that will solve the 'entire' surface problem.

[15]`https://www.zdnet.com/article/firefox-to-add-tor-browser-anti-fingerprinting-technique-called-letterboxing/`

5.1 System-, Browser- and User Behaviour

For each of the categories in the taxonomy, it is difficult to measure big each of the separate fingerprintable surfaces is. For system behaviour, browser behaviour and user behaviour, there is unlikely to be a solution to cover all aspects within a category. As all three require behavioural characteristics from either a system, a browser or a human, this can take on an enormous range of possible combinations. Another problem is that only anticipated behaviour can be detected, while new 'malicious' behaviour forms the biggest threat. As a partly solution for this in system behaviour, virtualization can be a solution. As introduced in Blink [LRB15b], the actual hardware behaviour will not be exposed to the fingerprinter. Problem is that the virtual machine itself is still fingerprintable and if not everyone uses the same virtual machine, this might at best reduce the chance to be uniquely identified.

A users behaviour is relatively unexplored in the current context of browser fingerprinting. Slightly touched upon by [SPK16] and not for the intended purpose of fingerprinting by [ML01] and [BBW17], it is hard to extract a pattern that matches a single person based upon behavioural characteristics. It could however be a good aid for current fingerprints to confirm the detection of bots, or as additional checks for verification of the visitor.

5.2 User Properties

User properties are gaining in popularity, as they introduce a lot of entropy. This properties can almost be seen as an infinite source of possible combinations, as there are very much plugins. It was shown that purely based on plugins and extensions [SAS17, SN17], it is possible to identify users or at least create a profile of them using the installed features. The same is true for custom fonts. As there will always be customization options for a browser, this part of the fingerprint surface is impossible to completely cover, making it a perfect fingerprintable surface.

5.3 System Properties

This can take on a lot of values, but is probably more limiting than the user properties. Looking for installed OS's, there are three 'big' vendors and when taking mobile into account this increases to five. Looking at certain vendors for CPU, the list is limited to a few settled companies and for GPU this is true as well. There is always a possibility that new operating systems enter the market or new CPU/GPU vendors will enter, but this is much more limited than the user properties, where a user installs plugins/extensions. More people can create simple different extensions, while not everyone can create custom made CPUs/GPUs.

5.4 Browser Properties

Browser properties are used in almost all studies, in combination with browser behaviour. This is mainly because of the widely available features and the ease to access them. This can *currently* be seen as the biggest fingerprint surface due the sheer amount of possible values and the easiest to gather via Javascript. Currently fingerprinters are taking a small fraction of the possible fingerprint in that area, but this can change. To understand how big the browser property category is, we introduce the Prop-test, a tool which iterates over all possible static values in a browsers `Window` instance and returns all values it can find. It covers the entire fingerprintable surface for the browser properties and shows what the exact fingerprint is. In the next chapter we will go into more detail about the Prop-test and why this covers the entire browser property surface.

5.5 Proof of concept implementation of Prop-test

The Prop-test is designed to display the whole fingerprint for the browser property category. There is not a tool present, which is able to show this entire surface[16] Most available tool like fingerprintJS2 or BlueCava only take a part of the browser properties and combine it with system properties, system behaviour, browser behaviour and user properties, making them incomplete at extracting a complete fingerprint. The Prop-test shows that is it possible to capture the entire category by looping over all possible variables within a current `window` object. From privacy perspective, the Prop-test shows all the variables which are easily acquirable for a website, meaning that if one wishes to be anonymous online, at least some of these values have to be changed. It can be either into a random value or into value which hides best into the entire crowd.

The Prop-test takes a top-to-bottom approach. It starts by parsing the `Window` object, sees what is inside and stores all these variables in a new array. It then does this again for the next layer, recursively, until a stop condition has been reached. Each layer returns all its variables and their values. As the `Window` object contains a lot of recursive elements, certain objects have to be filtered out to prevent an infinite loop. This filtered objects currently contain : `Window, HTMLCollection, HTMLhtmlElement, HTMLHeadElement`. This list can be extended if desired, or if new recursive elements appear in the future. There are additional (optional) filters which can be used to get better results e.g a filter to not show and process objects which do not have a child, or which only display a repetitive DOM e.g `innerHTML,outerHTML,outerText and textContent`. Using this approach, the smallest differences in the fingerprint can be found at the deepest layers. Currently the Prop-test is programmed to go four layers deep. In these four layers are at least all the variables which belong to the system properties category and are also checked by fingerprintJS. In the end, the Prop-test returns all the per layer variables and their values which are available to it. This makes the Prop-test not only a new fingerprinter, but could also be used in testing environments to test new counter measures, or look at the effectiveness of countermeasures in the system properties category.

6 Conclusion

We introduced a new way to reason about the fingerprint surface, as there seemed to be different terms used in multiple different papers. With this introduction we hope make reasoning about the browser fingerprint more clear and that programs in the future could focus better on certain aspects. We showed a systematization of knowledge from multiple papers over the last 9 years and showed what developments have been in the field and what directions are being explored, together with a categorization in our newly introduced system. We stated what needs to be done to reduce the fingerprint surface in the future from a more fundamental perspective, and argue that is might be impossible to capture the entire surface. We do however show a solution to capture one of the categories, the fingerprint surface for browser properties, and we explained its working. We hope to have given some handles which can be used to reason and discuss about the fingerprint surface, to improve upon techniques which are influenced by this.

7 Future work

Future work could into looking into making the Prop-test a more usable framework which can act as a testing tool for newly introduced plugins. In its current form it is very minimal and acts as a proof of concept. It can take multiple directions w.r.t defences against bots, testing new fingerprinters, or even use the Prop-test to extract a fingerprint itself using only browser properties, due to the high entropy.

[16]Disclaimer: the Prop-test acts as a proof of concept and might contain bugs.

References

[AEE$^+$14] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juárez, Arvind Narayanan, and Claudia Díaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 674–689, 2014.

[AJN$^+$13] Gunes Acar, Marc Juárez, Nick Nikiforakis, Claudia Díaz, Seda F. Gürses, Frank Piessens, and Bart Preneel. Fpdetective: dusting the web for fingerprinters. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 1129–1140, 2013.

[BBW17] Nicolas Van Balen, Christopher T. Ball, and Haining Wang. Analysis of targeted mouse movements for gender classification. *ICST Trans. Security Safety*, 4(11):e3, 2017.

[BFGI11] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. User tracking on the web via cross-browser fingerprinting. In *Information Security Technology for Applications - 16th Nordic Conference on Secure IT Systems, NordSec 2011, Tallinn, Estonia, October 26-28, 2011, Revised Selected Papers*, pages 31–46, 2011.

[BKST16] Peter Baumann, Stefan Katzenbeisser, Martin Stopczynski, and Erik Tews. Disguised chromium browser: Robust browser, flash and canvas fingerprinting protection. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society, WPES@CCS 2016, Vienna, Austria, October 24 - 28, 2016*, pages 37–46, 2016.

[Eck10] Peter Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies, 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings*, pages 1–18, 2010.

[Emb] O Emberton. `https://www.quora.com/Why-cant-bots-check-%E2%80%9CI-am-not-a-robot%E2%80%9D-checkboxes/answer/Oliver-Emberton?share=1`.

[EN16] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1388–1401, 2016.

[FE15] David Fifield and Serge Egelman. Fingerprinting web users through font metrics. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, pages 107–124, 2015.

[FvGJ18] Gertjan Franken, Tom van Goethem, and Wouter Joosen. Who left open the cookie jar? A comprehensive evaluation of third-party cookie policies. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, pages 151–168, 2018.

[GBLL18] A. Gómez-Boix, P. Laperdrix, and B. Laperdrix. Hiding in the crowd: an analysisof the effectiveness of browser fingerprinting at large scale. *WWW2018 - TheWebConf 2018 : 27th International World Wide Web Conference*, pages 1–10, April 2018.

[LBM17] Pierre Laperdrix, Benoit Baudry, and Vikas Mishra. Fprandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In *Engineering Secure Software and Systems - 9th International Symposium, ESSoS 2017, Bonn, Germany, July 3-5, 2017, Proceedings*, pages 97–114, 2017.

[LRB15a] P. Laperdrix, W. Rudametkin, and B. Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. *37th IEEE Symposium on Security and Privacy (S&P 2016). San Jose, United States.*, 2015.

[LRB15b] P. Laperdrix, W. Rudametkin, and B. Baudry. Mitigating browser fingerprint tracking: Multi-level reconfiguration and diversification. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 98–108, May 2015.

[LZC16] S. Luangmaneerote, E. Zaluska, and L. Carr. Survey of existing fingerprint countermeasures. In *2016 International Conference on Information Society (i-Society)*, pages 137–141, Oct 2016.

[MDM$^+$18] Pedro Marques, Zayani Dabbabi, Miruna-Mihaela Mironescu, Olivier Thonnard, Frances V. Buontempo, Ilir Gashi, and Alysson Bessani. Using diverse detectors for detecting malicious web scraping activity. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN Workshops 2018, Luxembourg, June 25-28, 2018*, pages 67–68, 2018.

[ML01] Florian Mueller and Andrea Lockerd. Cheese: tracking mouse movement activity on websites, a tool for user modeling. In *CHI '01 Extended Abstracts on Human Factors in Computing Systems, CHI Extended Abstracts '01, Seattle, Washington, USA, March 31 - April 5, 2001*, pages 279–280, 2001.

[MM12] Jonathan R. Mayer and John C. Mitchell. Third-party web tracking: Policy and technology. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 413–427, 2012.

[MRH$^+$12] Martin Mulazzani, Philipp Reschl, Markus Huber, Manuel Leithner, and Edgar R. Weippl. Fast and efficient browser identification with javascript engine fingerprinting technical report tr-sba-research-0512-01. 2012.

[MS12] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in html5, May 2012.

[NJL15] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. Privaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pages 820–830, 2015.

[NKJ$^+$13] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 541–555, 2013.

[NKJ$^+$16] Rishab Nithyanand, Sheharbano Khattak, Mobin Javed, Narseo Vallina-Rodriguez, Marjan Falahrastegar, Julia E. Powles, Emiliano De Cristofaro, Hamed Haddadi, and Steven J. Murdoch. Ad-blocking and counter blocking: A slice of the arms race. *CoRR*, abs/1605.05077, 2016.

[OACD15] Lukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Díaz. The leaking battery - A privacy analysis of the HTML5 battery status API. In *Data Privacy Management, and Security Assurance - 10th International Workshop, DPM 2015, and 4th International Workshop, QASA 2015, Vienna, Austria, September 21-22, 2015. Revised Selected Papers*, pages 254–263, 2015.

[SAS17] Alexander Sjösten, Steven Van Acker, and Andrei Sabelfeld. Discovering browser extensions via web accessible resources. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY 2017, Scottsdale, AZ, USA, March 22-24, 2017*, pages 329–336, 2017.

[SN17] Oleksii Starov and Nick Nikiforakis. XHOUND: quantifying the fingerprintability of browser extensions. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 941–956, 2017.

[SPK16] S. Sivakorn, J. Polakis, and A.D. Keromytis. I am robot:(deep) learning to break semantic image captchas, 2016.

[TJM15] Christof Ferreira Torres, Hugo L. Jonker, and Sjouke Mauw. Fp-block: Usable web privacy by controlling browser fingerprinting. In *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II*, pages 3–19, 2015.

[Vlo18] G. Vlot. Automated data extraction; what you see might not be what you get, 5 July, 2018.

[VRR18] Antoine Vastel, Walter Rudametkin, and Romain Rouvoy. FP -tester : Automated testing of browser fingerprint resilience. In *2018 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2018, London, United Kingdom, April 23-27, 2018*, pages 103–107, 2018.

[ZDLZ14] Zhe Zhou, Wenrui Diao, Xiangyu Liu, and Kehuan Zhang. Acoustic fingerprinting revisited: Generate stable device ID stealthily with inaudible sound. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 429–440, 2014.