Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bachelorarbeit

Implementation of browser fingerprinting detection and integration into PrivacyScore

vorgelegt von

Tronje Krabbe
geb. am 11. August 1994 in Hamburg
Matrikelnummer 6435002
Studiengang Informatik

eingereicht am 11. Mai 2018

Betreuer: Dipl.-Inf. Tobias Müller
Erstgutachter: Prof. Dr.-Ing. Hannes Federrath
Zweitgutachter: Prof. Dr. Dominik Herrmann

Abstract

Many websites use cookies or other means to track users across the web. This is often done for marketing or advertising purposes. Oftentimes, cookies are insufficient to reliably track a user, as they can be deleted by the user. Alternatively, to track a user, a fingerprint of their device and web browser can be constructed, which can identify the user uniquely within a large set of users.

This thesis aims to explore modern fingerprinting techniques, and to implement automated fingerprinting detection. This creates transparency for users who are concerned about their privacy, and who do not wish to have their online activity tracked and recorded.

We have explored related work on fingerprinting and fingerprinting detection, analysed many sites that use fingerprintable attributes or APIs manually, and implemented an automated fingerprinting detection method into PrivacyScore, a service that can scan and analyse a website in regards to its privacy and security.

Contents

1 Introduction

When browsing the web, users can be tracked through the use of cookies, which store information on the user's computer that can also be accessed by a remote server. If users wish not to be trackable, they can modify or delete any cookie as they see fit. There are, however, other methods of uniquely identifying users and tracking them during their use of the internet, which are not as easily mitigated as cookies [3]. One of these methods is called "browser fingerprinting", and it will be the focus of this thesis.

Browser fingerprinting, also known as device fingerprinting, and in the following often simply referred to as "fingerprinting", works by analyzing a web browser's configuration and settings; mostly installed fonts and HTML5 canvas behaviour [8], as well as language settings, time zone settings, installed add-ons, and more. Flash is also sometimes used, though its end-of-life lies in the foreseeable future, reportedly in the year 2020 [4]. Fingerprinting through Flash and the detection thereof will thus not be within the scope of this thesis.

The above attributes are readily available to be collected through JavaScript functions. Simply recording all function calls made by the JavaScript front end of a website can reveal whether fingerprinting is likely to be taking place or not [10, 11].

Techniques used to identify and track users across different websites without their knowledge or their ability to easily intervene, violates their privacy; by creating a fingerprint of a user's browser, with sufficiently sophisticated techniques, one can uniquely identify one user among hundreds of thousands [3], re-instate any cookies the user may have deleted, or simply track and analyse their use of any number of web services for a multitude of malicious reasons [7]. Methods exist to mitigate the effectiveness of browser fingerprinting, such as the one presented in [16]; these are, however, usually quite complicated and thus not suitable for ordinary users. We have therefore implemented a technique to recognize when a website is deploying browser fingerprinting, and along the way explored the common techniques used to construct a fingerprint. The resulting software has been integrated into PrivacyScore[1], a web-service to test and rank websites according to the extent to which they respect user privacy [17]. Informing users about websites they use and their treament of sensitive information can be seen as a different kind of defense against the violation of privacy that is fingerprinting.

1.1 Motivation

There is a fair amount of related work on the topic. In the fields of privacy and security research, we always deal with some form of 'adversary', meaning something or someone that can change and adapt to new research and methods. It is therefore important to keep up with any changes, which can happen very quickly in the context of software. We aim to analyse the current state of fingerprinting, and contribute to keeping up with modern privacy-violating techniques.

1. https://privacyscore.org

Tracking a user does not simply mean recognizing whether a visitor to one's website is a new or an existing user. Identifying information can be passed on or sold to partners, advertisers, or even governments, in order to construct a rich browsing history of a user. Users are required by EU law to be informed of cookies being set by websites, yet fingerprinting can and does happen quietly.[2] Our aim is to create more transparency about the websites that users visit.

1.2 Fundamentals

The following section will explain some fundamental knowledge that is referenced throughout this thesis.

1.2.1 Fingerprint

Throughout the thesis, terms such as "fingerprint", "browser fingerprint", etc. will be used largely interchangeably. A fingerprint, in most contexts, is something that can uniquely identify something else. The obvious example is a human's fingerprint, which is unique to each finger of each human, with very few exceptions, if any.[3]

1.2.2 Entropy

Entropy is the most important aspect of a fingerprint. Peter Eckersley describes entropy as "a mathematical quantity which allows us to measure how close a fact comes to revealing somebody's identity uniquely".[4] Entropy is often measured in bits, because it measures the different possible values something can take. To give an example, the outcome of a coin toss has one bit of entropy, as it can have two different values: heads or tails. A random, unknown human's identity contains about 33 bits of entropy, as two to the power of 33 is 8 billion, and there are some 7.6 billion humans alive at the time of writing. If a browser fingerprint can provide 33 bits of information, it can be used to uniquely identify any person, so long as all fingerprints are unique.

1.2.3 JavaScript

JavaScript[5] is, at the time of writing, the primary programming language understood by modern web browsers. While HTML and CSS dictate the visual layout and style of a web page, JavaScript provides its functionality. JavaScript is important in the context of this thesis, because it can be used to construct a browser fingerprint. It can query the attributes of a browser or device, transform them, and transmit them to a third party. Within this thesis, the JavaScript code executed by websites when visited, will be saved and analysed. This constitutes the primary source of data for the thesis.

2. http://ec.europa.eu/ipg/basics/legal/cookies/index_en.htm#section_2 Visited on May 3, 2018
3. https://www.interpol.int/INTERPOL-expertise/Forensics/Fingerprints
4. https://www.eff.org/deeplinks/2010/01/primer-information-theory-and-privacy
5. https://developer.mozilla.org/en-US/docs/Web/JavaScript

```
alert('Hello, world!');
```

Figure 1.1: A JavaScript "Hello, World" program

```
var _0xd533 = [
    "\x48\x65\x6C\x6C\x6F\x2C\x20\x77\x6F\x72\x6C\x64\x21"
];
alert(_0xd533[0]);
```

Figure 1.2: An obfuscated JavaScript "Hello, World" program

Obfuscation

JavaScript source code can be obfuscated. Obfuscation means to change the code until it is no longer possible for a human to read the code, at least not within a reasonable time frame. We have encountered many instances of obfuscated JavaScript code. For an example of JavaScript obfuscation, consider the code samples in Figure 1.1 and Figure 1.2. They are functionally equivalent. The code in Figure 1.2 has been obfuscated by JavaScript Obfuscator [12]. A more extreme example for obfuscation of JavaScript code is provided by JSFuck [13], the output of which cotains only six distinct characters, while still being technically valid JavaScript.

1.2.4 Canvas

The canvas [6] is a HTML5 element upon which a web page can draw, using one of several methods. For instance, WebGL allows rendering of complex 3D graphics on a canvas. The canvas can also be manipulated by JavaScript. For this thesis, the canvas plays an important role. It enables the generation of images with high entropy values in regards to their fingerprinting value. We will show that many websites draw on a canvas, extract the resultant raw pixel values from the canvas, and use it to contribute to the generation of a fingerprint. This is possible for a number of reasons. For one, a JavaScript program can instruct the browser it is running in to draw emoji on a canvas. If the browser supports emoji, they will be drawn in a certain way, usually dictated by the device the browser is running on. Apple's emoji look different from Google's emoji, for instance.[6][7] If emoji are not supported, fallback behaviour often means that black squares will be drawn in their place. Thus, drawing simple emoji on a canvas can give up information about a device's vendor. A similar approach can be taken with fonts. Creating a WebGL rendering context and rendering images on it can reveal information about the GPU and associated driver software of the host device.

6. https://emojipedia.org/apple/
7. https://emojipedia.org/google/

1.3 Goal and Leading Questions

The overall goal of this thesis is to create a software that can reliably report whether a website carries out fingerprinting, and to integrate it into PrivacyScore. To achieve this goal, we must inspect the state of browser fingerprinting across the web, and gain new insights into the topic. To this end, we will attempt to answer the following leading questions.

1. What are the most commonly used fingerprinting techniques and how can they be identified by our software?
Identifying the most common techniques is important, as it will allow the detection of the broadest spectrum of fingerprinters. What are they, and how can we best identify them?

2. How can false positives and false negatives be minimized?
False positives are, in our case, websites which we identify as fingerprinters, but which do not actually perform fingerprinting. False negatives, conversely, are sites that do construct fingerprints, but which we label as non-fingerprinters. Both are detrimental to the results of the software and thus must be minimized.

3. How do multiple different fingerprinting techniques relate?
We assume that a service provider is more likely to use more than one fingerprinting technique, and that the majority of fingerprinters also use the highest amount of different techniques. Is this the case? Which techniques are most often used together? Why?

1.4 Organization

The structure of this thesis is shown in the following. In Chapter 2, we present related work, split into two main groups: related work on the topic of browser fingerprinting and related work on browser fingerprinting detection. Chapter 3 will showcase our data acquisition methods and discuss our analyses and their results. In Chapter 4, we will give a technical overview of PrivacyScore, our algorithm, and the integration of our algorithm into PrivacyScore. We will describe and discuss our scoring algorithm. Finally, in Chapter 5, we will summarize our work and discuss possible future works.

2 Related Work

This chapter will give an overview of previous works. It is split into two sections. The first section presents existing work on the topic of fingerprinting. The second one will show existing work on the detection of fingerprinting.

2.1 Browser Fingerprinting

There are various closed-source implementations of browser/device fingerprinting. These are unavailable to us. To give an example, we will describe a script or family of scripts that are highly obfuscated, but also likely fingerprinters, in Section 3.4. However, we can turn to several open-source projects which implement fingerprinting for deeper insights into this technology, and to learn how we may be able to identify the closed implementations.

Essentially, fingerprinting works by combining data available to front-end JavaScript code into a string, usually by hashing some data structures. This string is the fingerprint. Different fingerprinting methods can be distinguished by the different attributes they use to construct their fingerprint.

For instance, one of the attributes providing the most entropy is acquired by canvas fingerprinting, which works by drawing several symbols like geometric shapes, emoji, and so on, on an HTML5 canvas element [15], as described in more detail in Section 1.2.4 The script in Appendix B.1 shows an example of canvas fingerprinting.

For a concrete example, *Am I Unique?* writes the same pangram[1] twice onto the canvas, using two different fonts, and each time followed by a special Unicode character, an emoji. Finally, a rectangle in a specific colour is drawn [15].

So, while different fingerprinting scripts are all likely to use canvas fingerprinting, they can differ in the exact method they use to create the canvas-related aspect of the fingerprint.

The methods described in this section can shed some light onto the different attributes used, and their computation, and how much entropy each can contribute to a fingerprint. We have compiled an overview of which attributes are used by which implementation in Table 2.1.

2.1.1 Am I Unique?

Am I Unique? [15] is a web service which creates a fingerprint with the press of a button.[2] Its purpose is to educate users about fingerprinting. *Am I Unique?* borrows part of its JavaScript fingerprinting code from Fingerprintjs2, which is discussed in Section 2.1.3.[3]

1. A sentence containing every letter of the alphabet.
2. https://amiunique.org
3. https://github.com/DIVERSIFY-project/amiunique/blob/master/website/public/javascripts/webGL.js#L2

2.1.2 Panopticlick

The EFF's Panopticlick project is similar to *Am I Unique?*, in that it aims to learn and inform about fingerprinting. It also offers a way to test any tracking protection a user may have enabled by simulating tracking domains [11]. The attributes used by Panopticlick to construct a fingerprint differ slightly from those used by *Am I Unique?*, though overall, they are similar.

Eckersley showed that the list of installed fonts and the list of installed plugins are the most identifying properties [7]. This finding, however, did not yet take canvas fingerprinting into consideration.

2.1.3 Fingerprintjs2

Fingerprintjs2 [21] is a popular JavaScript library that can be used to easily construct a fingerprint.[4] It can use a larger variety of attributes when compared to the two previously mentioned services by default. Which attributes are used to construct the fingerprint can be configured.

In Chapter 3, we will show that about 7% of sites we have analysed use Fingerprintjs2.

2.1.4 Fingerprint Central

Fingerprint Central, or FP Central, is a website which "aims at studying the diversity of browser fingerprints and providing developers with data to help them design good defenses" [14]. Much like Panopticlick or *Am I Unique?*, one can have the site generate a device fingerprint, and view the exact components of the fingerprint, as well as statistics about the fingerprints of other users. Of note is the fact that FP Central does not use canvas fingerprinting at this time. However, the service is still in its beta phase, so this may change in the future. Another interesting aspect of FP Central is the math fingerprinting. High-precision mathematical functions can have operating system specific behaviour, and thus provide entropy.[5] FP Central also performs audio context fingerprinting, which is also not present in any of the previously presented fingerprinters.

2.1.5 Study by Friedrich-Alexander-University

The Chair for Computer Science 1 of Friedrich-Alexander-University Erlangen-Nürnberg (FAU) is, at the time of writing, conducting a study about browser fingerprinting [5]. Their study has users sign up on their website, and then has users click links sent to them via email on a weekly basis. Upon visiting each link, a fingerprint of that user is generated and saved.

Their intermediate results show that 96% of their 33,302 collected fingerprints are unique.[6] These fingerprints were generated for 2197 registered participants during 43.2 weeks so far.

4. 4891 stars and 712 forks on GitHub. Visited on February 4, 2018
5. https://trac.torproject.org/projects/tor/ticket/13018
6. https://browser-fingerprint.cs.fau.de/statistics/ Visited on May 5, 2018

Table 2.1: Different Browser Attributes used by Fingerprinting Projects

Attribute	Am I Unique?	Panopticlick	Fingerprintjs2
User Agent	✓	✓	✓
Accept Header	✓	✓	-
Content Encoding	✓	-	-
Content Language	✓	-	-
List of plugins	✓	✓	✓
Cookies enabled?	✓	✓	-
Local storage available?	✓	-	✓
Session storage available?	✓	-	✓
Timezone	✓	✓	✓
Screen resolution	✓	✓	✓
Screen colour depth	✓	✓	✓
List of fonts	✓	✓	✓
List of HTTP headers	✓	-	-
Platform	✓	✓	✓
Do Not Track Header present?	✓	✓	✓
Canvas	✓	✓	✓
WebGL	✓	✓	✓
WebGL Vendor	✓	-	✓
WebGL Renderer	✓	✓	✓
Use of an ad blocker	✓	-	✓
JavaScript allowed?	-	✓	-
System Language	-	✓	✓
Touchscreen support	-	✓	✓
IndexedDB available?	-	-	✓
Open DB?	-	-	✓
CPU class	-	-	✓
Available processors	-	-	✓
Device memory	-	-	✓

One very interesting takeaway from their intermediate results is that 69% of all *userAgent* strings are unique amongst their participants. This makes the *userAgent* a prime attribute to include in a fingerprint, and can be used by us in our implementation; for a website where the fingerprinting detection is otherwise inconclusive, the use of the *userAgent* can be the deciding factor.

Their fingerprint seems to include browser and operating system, whether cookies, do-not-track, JavaScript, Flash, Java and/or Silverlight are enabled, number of plugins, number of MIME types, and the number of installed fonts. This is based on the information on their statistics page. A detailed description of their fingerprinting technique is absent on their website.

2.2 Browser Fingerprinting Detection

At a basic level, in order to detect fingerprinting, all JavaScript calls to the relevant attributes, like User Agent, HTML5CanvasElement, and so on, need to be recorded. Then, a decision can be made about whether the number of calls is suspicious, or constitutes normal website behaviour.

However, since these attributes also have legitimate uses, this is not a trivial task. This section will present previous work on this topic.

2.2.1 FPDetective

Acar et al. provide some useful metrics to differentiate between fingerprinting and legitimate use of browser attributes [1]. They have implemented a fingerprinting-detection framework called *FPDetective*. *FPDetective* focuses on font-based fingerprinting, and uses *PhantomJS* to log JavaScript function calls and *Chromium* to log Flash actions. Much like OpenWPM, their crawler logs accesses to certain browser and device properties, such as those contained within the *window.navigator* object. Popular attributes like the HTML5 Canvas and WebGL were ignored in the study and by the framework's logging.

Partly automated, their process consisted of an automated analysis in regards to font fingerprinting, and a manual analysis of JavaScript and decompiled Flash code. Their focus on fonts derives from the assumption that a website that enumerates fonts is a likely fingerprinter. *FPDetective* classifies a script as a fingerprinter if "it loads more than 30 system fonts, enumerates plugins or mimeTypes, detects screen and navigator properties, and sends the collected data back to a remote server" [1].

As Flash is nearing its end-of-life, this thesis opts not to analyse Flash programs delivered by websites, which means *FPDetective*'s extensive work on Flash-based fingerprinting is of little use for us.

2.2.2 OpenWPM

Englehardt et al. have developed a framework called *OpenWPM*, which can be used to analyse the way a website handles user privacy. They have used this to analyse the tracking behaviour of one million websites [8, 9].

Englehardt et al. differentiate between distinct forms of fingerprinting. They classify a script as a canvas-fingerprinter, if there is a canvas larger than 16 by 16 pixels, and text is written to it in either at least two different colors, or with 10 distinct characters. A canvas-fingerprinter script should *not* call the *save*, *restore* or *addEventListener* methods of the rendering context. They have not further justified why the calling of these methods is a criterium to label a script non-fingerprinting. We assume that these methods suggest legitimate use of the API. However, when analysing a script automatically, we cannot take the absence of these methods into consideration. A script looking to evade our detection could simply call these methods without using them for any practical reason.

Finally, the script has to call either *toDataURL* or *getImageData* for an area of at least 16 pixels in width and height. They have manually verified the accuracy of their heuristic, and found only four false positives out of 3493 scripts. We will show that calling *toDataURL* on a canvas element is indeed a good indicator of fingerprinting in Section 3.3.

Englehardt et al. make a distinction between canvas fingerprinting and canvas font fingerprinting. They classify a script as a canvas font fingerprinter if it sets the *font* property of a canvas at least 50 times, and also calls the *measureText* method at least 50 times. They have manually verified that there were no false positives for this metric.

Their findings show that only about 1.6% of the analysed sites employ canvas-based fingerprinting. However, they also show that among the top 1,000 sites, 5.1% of sites employ canvas-based fingerprinting.

Their work also examines new, previously unknown methods of fingerprinting, such as canvas font fingerprinting, which works by rendering text in a large number of different fonts on a canvas element; WebRTC-based fingerprinting, which, as the name suggests, abuses the WebRTC framework; as well as AudioContext fingerprinting, and Battery API fingerprinting.

OpenWPM uses Firefox, Selenium, and custom JavaScript injected into loaded sites to log calls to relevant browser attributes. It constitutes the technological basis of this thesis' work, as it allows easy crawling of websites, while recording most if not all relevant information.

2.2.3 Tor Browser

The Design and Implementation of the Tor Browser [DRAFT] states that "[the authors] believe that the HTML5 Canvas is the single largest fingerprinting threat browsers face today", as a canvas offers easy and high-entropy fingerprinting [2, 18, 20]. The Tor Browser can alert users to data collection from a HTML5 Canvas element. Data collection, in this context, means using a function like *toDataURL* or *measureText* on the element.

3 Data Acquisition and Analysis

This chapter presents the dataset used to conduct the experiments for this thesis. We will describe how the data was acquired, and present an analysis of the fingerprinting behaviour contained within.

Our goal is to base a scoring algorithm on our findings, which we will integrate into PrivacyScore. A good scoring algorithm will distinguish at least three cases. That a site is very likely fingerprinting, that a site is very likely not fingerprinting, or that there is insufficient data to make a meaningful decision. PrivacyScore has a rating system that distinguishes between the values *critical*, *bad*, *warning*, *neutral*, *good*, and *doubleplusgood*.[1] Our scoring algorithm will need to map its results to these ratings. Obviously, a site that uses no fingerprintable attributes or APIs will get a *doubleplusgood* rating; the more attributes and APIs are used, the lower the rating. We will present our algorithm in Chapter 4.

3.1 Acquiring Site Information

As described in Section 2.2.2, we have used OpenWPM[2] to gather data about website behaviour. The parameters used with OpenWPM to gather the data can be found in Appendix A.

OpenWPM saves most data in SQLite3 databases.[3] This includes the data interesting to us: JavaScript calls, function names, and script URLs. We analysed the SQLite3 databases using Python3 scripts. Physical distributions of this thesis include the source code on a USB drive. We will show some of the SQL queries we used to extract interesting data throughout Chapter 3.

Alexa regularly compiles a list of the top one million websites, ordered by the number of visitors.[4] We have analysed the top 5000 websites, as described by a version of this list acquired from http://s3.amazonaws.com/alexa-static/top-1m.csv.zip on December 19, 2017.

Multiple datasets based on this list were gathered. Due to timeouts or other errors, not all 5000 sites were actually crawled each time. The primary dataset we used was crawled on January 27, 2018, and includes 4929 sites with distinct URLs. Some of these sites may be duplicates of each other. For instance, the URL https://fbcdn.net/ for Facebook's content delivery network simply points to https://facebook.com/, both of which are included in the dataset.

1. https://github.com/PrivacyScore/PrivacyScore/blob/master/privacyscore/evaluation/rating.py#L11 Visited on May 3, 2018
2. https://github.com/citp/OpenWPM
3. https://www.sqlite.org/index.html
4. https://www.alexa.com/topsites

We make the distinction between the primary dataset and secondary datasets. We have run all analyses on the primary dataset, and we will show that the noted behaviour is consistent by repeating some analyses on the secondary sets. We do this because running all analyses on all sets is beyond the time constraints of this work. In the following, we will refer to the primary dataset as simply "the dataset".

3.2 Data Properties

In this section, we will give an overview of basic properties of the primary dataset, with special regard to their relevance for fingerprinting. Table 3.1 shows some of these properties.

The first block of Table 3.1, titled *Basic attribute lookups*, shows how many sites looked up certain browser or device attributes. We label these as "basic", because a JavaScript program can access them with a single function call. In this section of the table, we can see that, for example, 90% of sites in the set access the *userAgent* property. How many websites access which attribute is interesting to us because of the basic attributes' ease of access. As their values are easy to acquire, one might expect that these attributes are consistently used by different fingerprinters.

The next section in the table shows how many sites use more than one of the basic attributes. Note that the rows are not mutually exclusive. This means that, e.g. the 4245 sites using "at least" three attributes also contain all sites that use more than three attributes. This information is useful, because fingerprints are usually constructed from multiple data points, so that they are sufficiently unique [11].

The final block of Table 3.1 is focused on the HTML5 Canvas element. More specifically, we show how many sites use any kind of Canvas-related JavaScript, as well as how many use the specific data extraction methods. A website qualifies as using "any Canvas-related JavaScript" when it makes use of any function calls that are part of the Canvas API.[5] The two methods that can be used for data extraction are *toDataURL*[6] and *getImageData*[7]. This part of the table is of interest, because the Canvas element is one of the most useful elements for the construction of a fingerprint (cf. Sec. 1.2.4).

Of the 884 sites that use *toDataURL* or *getImageData* on a HTML5 Canvas object, which suggests a high likelihood of canvas fingerprinting, 536, or about 61%, also use eight or more of the basic attributes. These 536 sites (not shown in the table) are good candidates for likely fingerprinters. And indeed, 147 of them are contained in the 336 sites that we have identified as users of *Fingerprintjs2*. *Fingerprintjs2*, as discussed in Section 2.1.3, can be configured to include any number of attributes in its generated fingerprint.

From the above, we can also conclude that 189 users of *Fingerprintjs2* do not use either of the data extraction methods of the Canvas element. This is curious because of the high entropy of the Canvas fingerprinting technique. The cause for this could be that the sites are trying to avoid detection, or that they do not want to alert their users. The Tor Browser can alert users when a site is trying to extract data from a canvas (cf. Sect. 2.2.3).

5. https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API
6. https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/toDataURL
7. https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/getImageData

Table 3.1: Behaviour of Analysed Websites

	amount	percentage
All sites	4929	100%
Basic attribute lookups		
userAgent	4424	90%
language or languages	3985	81%
colorDepth	3931	80%
localStorage	3772	77%
platform	3275	66%
sessionStorage	2860	58%
cookieEnabled	2823	57%
doNotTrack	1406	29%
oscpu	580	12%
Using multiple of above attributes		
at least 2	4338	88%
at least 3	4245	86%
at least 4	3968	81%
at least 5	3572	72%
at least 6	2906	59%
at least 7	1963	40%
at least 8	1138	23%
9 attributes	473	10%
Canvas-related functions or attributes		
Using any Canvas-related JavaScript	1657	34%
toDataURL or getImageData	884	18%
toDataURL	768	16%
getImageData	252	5%
Calling functions defined by *Fingerprintjs2*	336	7%

```
SELECT DISTINCT script_url
FROM javascript
WHERE symbol = 'HTMLCanvasElement.toDataURL';
```

Figure 3.1: SQL query to select all JavaScript programs using *toDataURL*.

The number of users of Fingerprintjs2 was found by simply querying the database for function names that are consistent with those used in Fingerprintjs2, and therefore do not include sites which use an obfuscated Fingerprintjs2. Fingerprintjs2 has a few functions starting with the prefix *getHasLied*, like *getHasLiedOs*, the purpose of which is to report whether a browser tampered with certain information, for instance in the interest of mitigating fingerprinting.[8] Due to the peculiar name choice, and the fact that the same sites that call one *getHasLied* function also call all others, we can be quite certain that they all include Fingerprintjs2.

3.3 *toDataURL* Usage

In Section 3.2 we suggested that a use of *toDataURL* suggests a high likelihood of canvas fingerprinting. To show the plausibility of this claim, we have performed a manual analysis of the scripts served by 75 sites that use *toDataURL*. These 75 sites were randomly sampled from the list of sites which call *toDataURL*, but have not been identified as users of FingerprintJS2. We exclude users of Fingerprintjs2, as they would not provide any further insights, and we already have a good estimate of how many sites in our dataset use it, as shown in Table 3.1.

OpenWPM gives us a SQLite database, which includes a *javascript* table. This table includes a column called *script_url*, which contains the URL to a JavaScript file, and a column called *symbol*, containing the name of the symbol accessed by a JavaScript program. Using the query depicted in Figure 3.1, we can find all scripts that access the symbol *HTMLCanvasElement.toDataURL*.

The analysis of these scripts was done by looking for tell-tale signs of fingerprinting in the (potentially minified) JavaScript code. These signs are enumeration of fonts, writing to canvases, never displaying canvases, enumeration of plugins, and suspicious function names like *getFingerprint*.

Following the analysis, we claim with confidence that 51 out of the 75 sites are fingerprinters. Table 3.2 shows more details of this analysis.

The first block of Table 3.2 shows the sites we have deemed to be fingerprinters. Of these, 26 use a popular third-party script. This is interesting, because the behaviour of widely used scripts can be used to tune our scoring system. However, when assigning a score, looking for suspicious scripts based on name or URL alone, is not practical. This is because a fingerprinter can hide this from us easily, if they wanted to. One way to hide a script is to obfuscate it instead of simply minifying. Another way to hide its origin is to serve it as

8. https://github.com/Valve/fingerprintjs2/wiki/Browser-tampering Visited on February 8, 2018

Table 3.2: Behaviour of *toDataURL* Users

	amount	percentage
analysed sites	75	100%
Fingerprinting sites	51	67%
Using a popular third-party script	26	34%
Using functions with suspicious name	11	15%
Using a popular pangram on canvas	4	5%
Inconclusive	16	21%
Legitimate users	9	12%
Emoji-detection	7	9%
Applying blur to image	1	1%
Lazily loading an image	1	1%

Figure 3.2: Pangram with emoji as used by *Am I Unique?*, among others

a first-party script, not a third-party script, and replacing its name in a URL with random letters or words.

The block showing the fingerprinting sites also shows that 11 of them use a suspicious name for a function. To give some examples: *getCanvasFp*, *getCanvasPrint*, *canvasFingerprint*. We assume that any function would not be named in such a way if it did not indeed perform fingerprinting. The scripts using the suspicious names additionally showed other behaviour exhibiting the signs of fingerprinting. Looking for suspicious function names while scoring should be avoided, as adversaries can simply change the function names to avoid or limit detection, as many scripts already do through obfuscation.

Four of the found fingerprinters used the popular pangram shown in Figure 3.2 to perform canvas fingerprinting. Unfortunately, this is not useful in scoring future fingerprinters, as they can easily change the pangram they use.

We label sixteen of the *toDataURL* users as inconclusive. Their JavaScript was too obfuscated to draw meaningful conclusions. While obfuscation may itself be suspicious, many sites could perform it simply to keep their codebase closed-source.

The final block of Table 3.2 shows sites which made "legitimate" use of *toDataURL*. Seven sites used a canvas to detect how well emoji were supported. This is done by drawing emoji to a canvas, and analysing the array returned from *toDataURL*. If emoji are not supported by the system, they will be drawn as black squares or other non-emoji symbols.

One site used *toDataURL* seemingly to apply blur to an image. The final entry in Table 3.2 shows one site using a JavaScript module or library seemingly called "lazyimage" to lazily

```
SELECT DISTINCT site_visits.site_url
FROM site_visits
  JOIN javascript
    ON site_visits.visit_id = javascript.visit_id
WHERE javascript.symbol LIKE '%audio%'
   OR javascript.symbol LIKE 'AnalyserNode%'
   OR javascript.symbol LIKE 'GainNode%'
   OR javascript.symbol LIKE 'OscillatorNode%'
   OR javascript.symbol LIKE 'ScriptProcessorNode%';
```

Figure 3.3: Query to select all users of audio-related JavaScript calls

Table 3.3: Behaviour of Audio API Users

	amount	percentage
analysed sites	64	100%
Fingerprinting sites	46	72%
PerimeterX script users	15	23%
Facebook script users	13	20%
Inconclusive	17	27%
users of too obfuscated b2c script	11	17%
otherwise too obfuscated	5	8%
otherwise inconclusive	1	2%
Legitimate users	1	2%

load an image. This means to include a placeholder image on the site, and only load the actual image when needed.

3.4 Audio Fingerprinting

The distinct JavaScript interfaces related to audio present in our dataset, as recorded by OpenWPM, are: *AudioContext*, *OfflineAudioContext*, *AnalyserNode*, *GainNode* and *ScriptProcessorNode*.[9]

The SQL query in Figure 3.3 selects all sites that use a JavaScript interface or symbol related to audio. It returns 64 rows when executed on our dataset, meaning there are 64 potential audio fingerprinters, which represents about 1.2% of our set. We have performed a manual analysis of these sites in regards to audio fingerprinting similar to the one we have performed for *toDataURL*. The results are displayed in Table 3.3.

9. See also: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API

The most important takeaway from Table 3.3 could be that we were able to find just one legitimate use of the JavaScript Audio API. "Legitimate" in this case means that it is used to generate and play audible sounds for a visitor to hear.

Given the high amount of "inconclusive" entries, we have visited each Audio user labeled as such in a Chromium web browser, which produces the version string "Version 65.0.3325.181 (Developer Build) (64-bit)". This browser was otherwise unmodified, meaning there were no plugins, such as ad-blockers, installed. Note that this version of the Chromium web browser supports the Audio API, which we have manually verified by creating an instance of *AudioContext* in the console of the developer tools.

We have visited each site and waited for it to finish loading completely, and waited a further 10 seconds, exactly like OpenWPM in our automated data collection. One of these sites has played audio, although via an embedded YouTube livestream. This, as far as we could tell, does not use the Audio API.

Thus, we are confident that our dataset indeed contains only a single legitimate use of the Audio API.

Table 3.3 further shows the use of three prominent, reoccuring scripts, the most used of which is one provided by PerimeterX[10]. PerimeterX offers "Bot Detection and Anti Bot Protection with Unparalleled Accuracy", according to their website's home page. To this end, they seem to construct a browser fingerprint. Their script accesses numerous properties of the *window.navigator* object, uses *toDataURL*, *WebGL*, the Web Audio API, and enumerates 32 different fonts.

The second most used script originates from Facebook. It can be found under the same URL for each of the 13 sites in our set which use it.[11] Along with the Web Audio API, it uses the Canvas API, enumerates 26 distinct fonts and accesses other properties of the *window* and *window.navigator* objects which are often used in fingerprinting. We have also found the "mmmmmmmmmmlli" magic string in the JavaScript code, which we will further discuss in Section 3.5. Thus, we also classify it as a fingerprinter.

We refer to the final script as "the b2c script", although it may in fact be more than one distinct script. We have observed it to be served from mutliple different URLs, all beginning with *api.b2c.com/api/init-*, and ending with a mix of letters and numbers and the *.js* extension. All of these scripts were much too obfuscated to make any sense of. We have thus labeled their behaviour "inconclusive". However, we assert that it is very likely that these scripts are indeed fingerprinters. OpenWPM has recorded 31 distinct symbols being accessed by these "b2c" scripts, among them those related to the Canvas API, Web Audio API, WebGL, *localStorage*, *sessionStorage*, as well as multiple "basic properties".

We conclude that the Audio API is a prime indicator of browser fingerprinting, as the instances of fingerprinting use far outweigh its use of actual audio playback.

10. https://www.perimeterx.com/
11. https://cdn.atlassbx.com/FB/11122200772940/browser_features1488235112.js, visited on March 30, 2018

3.5 Font Fingerprinting

Font fingerprinting is easy to recognize when analysing minified JavaScript, as it works by enumerating various fonts to see if they are available on the fingerprinted system. This can be done by writing text to a Canvas in various fonts and measuring the rendered writing using the *measureText* method [9].

Because of its usual use of the Canvas element, it is closely related to Canvas fingerprinting, and it can be difficult to classify in some cases. In Section 3.3 we have shown an example of a "magic string" pangram, which also contains some emoji. It is used by *Am I Unique?* and others to fingerprint via the Canvas element. To that end, it is written onto a canvas in two different fonts, and some other elements are drawn as well, and the pixel data is then extracted. This method obviously uses font information to improve the fingerprint. However, other methods are more font-centric and use many more fonts, as well as the *measureText* method. Yet other methods do not use a Canvas element at all.

Font fingerprinting in the form where many fonts are enumerated is easy to spot in minified JavaScript, as there is usually an array containing many font names present. For automated scoring, counting how many fonts were set on a Canvas element can reveal Canvas-based font fingerprinting. For non-Canvas based font enumeration, the property accessors *offsetWidth* and *offsetHeight* are used, the use of which we can also count. This is described in greater detail below.

Within the context of Font fingerprinting, we have identified the "magic string" *mmmmmmmmmmlli*. To the best of our knowledge, we are the first to notice this string in relation to fingerprinting. It is used by at least ten sites in our dataset, and always occurs with exactly ten times the letter "m", two times the letter "l", and one letter "i".

We are unsure about the real number of sites in our dataset that use this magic string, because it is sometimes used within a *span* element instead of a canvas. JavaScript calls on this type of element are not instrumented by OpenWPM. Thus, there may be more sites that use this magic string to fingerprint in our dataset. We have included an example of this type of fingerprinting in Figure C.1 in the Appendix. The *mmmmmmmmmmlli* string is also present and used in the source code of Fingerprintjs2, though, since it is configurable, it does not reveal much more information about how many websites make use of the string. It does suggest, though, that a large number of websites indeed make use of it.

We assume that the *mmmmmmmmmmlli* string has certain subtle differences in width when rendered by different browsers or font-rendering libraries on different systems, with different fonts or font-families. We leave the determintation of the exact reason to use this particular string for future work.

Nikiforakis, et al. describe an observed font fingerprinting technique that works by setting a text within a *span* HTML element, and measuring its width and height after setting different fonts [19]. If a measurement for a certain font is different from that of a default fallback font, the fingerprinter can conclude that that particular font is installed.

Table 3.4: Results of secondar analysis regarding font fingerprinting.

	amount	percentage
Analysed sites	75	100%
Fingerprinting sites	5	7%
Fingerprintjs2	3	4%
PerimeterX	1	1%
Other	1	1%
Too obfuscated	7	9 %
No suspicious behavior	63	84%

3.5.1 Secondary Crawl Results

The specific JavaScript methods used for non-canvas-based font fingerprinting are called *offsetWidth* and *offsetHeight*, and are, again, not instrumented by OpenWPM. We have added the required instrumentation into the OpenWPM instance used by PrivacyScore for our implementation. To test the instrumentation, we have performed a secondary crawl of the top 500 sites in our list (cf. Sec. 3.1) with this instrumentation enabled.

Of these 500 sites, 389 use both *offsetWidth* and *offsetHeight* on any HTMLElement. Because of the high fraction of sites that use these methods among the set, we don't simply pick a certain number of sites to analyse. We narrow down the likely fingerprinters by selecting only those sites that call the methods at least 30 times. We base this number on the findings of FPDetective (cf. Sec. 2.2.1). We are left with 219 sites that call either or both of *offsetWidth* and *offsetHeight* at least 30 times. Of these 219 sites, we choose 75 at random and analyse their behavior further. We look for arrays of font names in the code, as well as the known magic strings, signs of other types of fingerprinting, and suspicious function names.

The results are presented in Table 3.4. The vast majority of analysed sites shows no suspicious use of the *offsetWidth* and *offsetHeight* properties. That does not mean they do not fingerprint, but that they do not use these properties to fingerprint. Five sites have been identified as fingerprinters, three of which use the Fingerprintjs2 script (cf. Sec. 2.1.3), one uses a script by PerimeterX, which we have already mentioned in Section 3.4, and one seems to use a different solution. Seven sites were too obfuscated to make any sense of.

We conclude that while the discussed properties can be used for fingerprinting, they are not very useful to incorporate into our scoring. Too many websites access these properties too many times, making their presence not especially suspicious.

3.6 WebGL Fingerprinting

The results of our analysis, as displayed in Table 3.5, again show that this API is used by many fingerprinters in our dataset. With WebGL, there is a higher number of sites which we have labeled "inconclusive". This is largely because many were too obfuscated to make sense of.

```
SELECT DISTINCT site_url
FROM site_visits
  JOIN javascript
    ON site_visits.visit_id = javascript.visit_id
WHERE symbol LIKE '%getcontext%'
  AND arguments LIKE '%webgl%';
```

Figure 3.4: Query to select all users of a WebGL context

Table 3.5: Behaviour of WebGL Users

	amount	percentage
Analysed sites	75	100%
Fingerprinting sites	34	45%
Inconclusive	21	28%
too obfuscated	10	13%
otherwise inconclusive	11	15%
Legitimate users	20	27%

Examples of legitimate use of the API include pbskids.org, which uses WebGL for a 2D game engine called PixiJS.[12] Some other sites seem to use a media player with this API, like geniuskitchen.com, which uses a player by Anvato.[13] One site, emol.com, uses *videojs-panorama*, a library to create a 360°video.[14]

As OpenWPM in its current version does not instrument calls to the WebGL API, we have to work around this by using the SQL query shown in Figure 3.4. There are no other symbols that OpenWPM instruments that are related to WebGL.

This also limits our methods of automatically labeling a script a fingerprinter. The only information we can gather from the database OpenWPM creates is whether a script creates a WebGL context or not.

We have patched OpenWPM to be able to instrument the WebGL API. Specifically, we can instrument the *WebGLRenderingContext* object, and have performed another crawl of the top 500 sites of our list (cf. Sec. 3.1), while instrumenting specifically the method *getExtension* of the *WebGLRenderingContext* object. This method, when given the parameter *WEBGL_debug_renderer_info*, produces another object which reveals information about the vendor and model of the GPU of the host computer.[15] This object does not hold any other information, and cannot be used for any purpose other than finding out the GPU vendor and

12. http://www.pixijs.com/ Visited on May 8, 2018
13. https://www.anvato.com/products/players/ Visited on May 8, 2018
14. https://github.com/yanwsh/videojs-panorama Visited on May 8, 2018
15. https://developer.mozilla.org/en-US/docs/Web/API/WEBGL_debug_renderer_info

```
var canvas = document.createElement("canvas");
var gl = canvas.getContext("webgl");
db_info = gl.getExtension('WEBGL_debug_renderer_info');
console.log(gl.getParameter(db_info.UNMASKED_VENDOR_WEBGL));
console.log(gl.getParameter(db_info.UNMASKED_RENDERER_WEBGL));
```

Example console output:
```
NVIDIA Corporation
GeForce GTX 970/PCIe/SSE2
```

Figure 3.5: JavaScript code that reveals GPU model and vendor.

```
SELECT DISTINCT site_url
FROM site_visits
  JOIN javascript
    ON site_visits.visit_id = javascript.visit_id
WHERE symbol LIKE 'RTCPeerConnection%';
```

Figure 3.6: Query to select all users of the WebRTC API

model. The code in Figure 3.5 shows an example of the fingerprintable information revealed by this object.

Of the 500 sites in the additional crawl, 22 used the method *getExtension* with the parameter *WEBGL_debug_renderer_info*. Of these 22, eight used Fingerprintjs2 to fingerprint, four used a script by PerimeterX to fingerprint, five fingerprinted otherwise, and the remaining five were either too obfuscated or did not fingerprint.

We can conclude that the above behaviour that can be used to find out the GPU vendor and model is a good indicator for fingerprinting, although our results are not based on as strong a dataset as other results presented by this thesis. Still, we contend that fetching this GPU information is suspicious enough to warrant a more negative rating in our scoring scheme.

3.7 WebRTC Fingerprinting

Our analysis of WebRTC users shows similar results as our analysis of WebGL users. Of the 75 sites we have analysed, we have labeled 15 as definite fingerprinters. Nine sites were labeled as legitimate users of the WebRTC API. The remaining 51 are inconclusive. Of these 51, 14 were too obfuscated to make sense of, and most, 37, used the same script served from the URL https://static.adsafeprotected.com/sca.17.4.20.js. We believe that this script enumerates browser features, based on many function definitions which try to access some web API and return a boolean value representing either success or failure to access that API. However, we are not certain that it constructs a fingerprint in the traditional sense. In

Table 3.6: Behaviour of WebRTC Users

	amount	percentage
analysed sites	75	100%
Fingerprinting sites	15	20%
Inconclusive	51	68%
using adsafeprotected script	37	49%
too obfuscated	14	19%
otherwise inconclusive	1	1%
Legitimate users	9	12%

Table 3.7: Numbers of Fingerprinting APIs Used

APIs used	amount	percentage
analysed sites	60	100%
1	17	23%
2	21	28%
3	26	35%
4	9	12%
5	2	3%

fact, the script generates a hash value that is accessible from the JavaScript console of the developer tools of a browser, and this hash changes between reloads of the site, suggesting no strictly fingerprinting behavior. We stress, though, that we do not fully understand this script, and that it does access and somehow process fingerprintable system information.

Legitimate users of the WebRTC API include:

- redbus.in, which uses a script served by gsecondgreen.com. The latter of the two offers a "customer engangement platform", and seems to use WebRTC to send various information to a remote server. However, as far as we could tell, this does not include any kind of fingerprint.
- stripchat.com, which uses Fingerprintjs2 to fingerprint, seems to use WebRTC legitimately to stream video and audio content from their performers to end users. Their script is also used by xhamsterlive.com.
- livejasmin.com, which uses WebRTC to stream video and audio as well.

Table 3.8: Extrapolated Frequency of Fingerprinting Techniques

	Fingerprinters	Total API Use	Estimated Fingerprinters
Audio	72%	64	1%
Canvas	67%	768	10%
WebGL	45%	438	4%
WebRTC	20%	281	1%

3.8 Multiple Fingerprinting Techniques

We assume that a site which was identified to fingerprint using one method is likely to also be using another method. This is a reasonable assumption, as more techniques give more entropy, and more entropy makes a more unique and therefore better fingerprint.

To substantiate our claim, we have analysed 75 sites which we have previously identified as fingerprinters. These sites were picked randomly from our set of already identified fingerprinters.

We exclude the basic attributes, such as the *userAgent*, when we count the methods used by a site. The counted methods are thus Audio-, Canvas-, Font-, WebGL- and WebRTC-Fingerprinting. All analysed sites also used basic attributes to enhance their fingerprint.

The results are shown in Table 3.7, and support our assumption. About a quarter of the fingerprinters use one API to fingerprint, and the remainder use more than one. Between one and three APIs used are the most common cases, four and especially five different APIs are much less common.

We conclude that it is not especially rare for a site to use only one fingerprintable API in addition to the basic browser attributes, but the majority of fingerprinters use more than one such API.

3.9 Most Common Techniques

Based on our findings, we can extrapolate an estimate for the most commonly used fingerprinting techniques. Table 3.8 shows basic estimations of the total fingerprinters for each technqiue. The "Fingerprinters" column shows how many sites of the subset we have analysed for that technique were deemed fingerprinters. The column "Total API Use" shows, how many sites in our dataset use the fingerprintable API (in any way). If we apply the first percentage to the number of that column, we get an estimate of how many sites in our dataset use this API to fingerprint. The column "Estimated Fingerprinters" shows the resulting percentage.

For example, 67% of the 75 sites that use the Canvas API we have analysed are fingerprinters. The total number of sites that use the Canvas API in our entire dataset is 768. That means that the total number of sites that fingerprint with this API in our dataset can be estimated as $67\% \cdot 768 \approx 515$. 515 sites out of the total 4929 is roughly 10%. That means we can estimate that 10% of all sites in our dataset use Canvas fingerprinting.

We can see that Canvas fingerprinting is by far the most popular, though the Audio API, when used, is more likely to be used for fingerprinting. With this, we have answered our first leading question (cf. Sec. 1.3).

3.10 Discussion

In the following, we will discuss the acquired data and the analyses we have performed.

3.10.1 Fingerprinters

The data shows us that simply by using a Canvas object a certain way, or by using the Audio API, a site is already a likely fingerprinter. We can assert this, because our analyses have shown 67% of analysed users of *toDataURL*, 72% of analysed Audio API users and 45% of analysed users of WebGL to be fingerprinters. Furthermore, 20% of WebRTC users fingerprint, in addition to any fingerprinters we were unable to identify, like the 49% that use a suspicious but inconclusively labeled script. Thus, just by using one or more of the listed APIs, we ascertain that a website is more likely to be a fingerprinter than not.

We have also found certain giveaways of sites being fingerprinters. The *mmmmmmmmmmlli* magic string is used exclusively by font fingerprinting scripts. The pangram *Cwm fjordbank glyphs vext quiz* followed by an emoji is similarly used often and only by fingerprinters. Oftentimes, fingerprinter scripts contain very suspicious function names, such as *getCanvasFingerprint*. While the absence of these details does not help in classifying a site, their presence can be used to instantly classify a site or script as a fingerprinter.

3.10.2 Limitations

When analysing a site, it is difficult or even impossible to tell whether an executed script sends data back to the site's servers, or a third party. To give an example, many sites include some kind of "share" button, oftentimes for social media platforms like Facebook or Twitter. These buttons are not always simply images surrounded by an HTML *a*-tag, but include JavaScript code as well. This JavaScript, we assume, communicates with the network which "owns" the button, not the site where the button is displayed. And this JavaScript may be a fingerprinter.

From a user's perspective, however, there may not be much of a difference. In the end, the user is still getting fingerprinted and tracked, regardless of who is at fault.

4 Design and Implementation

In Chapter 3, we have shown many users of certain JavaScript APIs to be fingerprinters. We can build upon this knowledge to score websites in regards to their likelihood to be fingerprinting.

Our approach to implement automated fingerprinting detection is described in the following. The software will analyse the behaviour of a website based on a data gathering by OpenWPM. We will extract all information relevant to fingerprinting from the database, and present this to the user of the detection software. We will highlight any giveaways of fingerprinting, such as included magic strings or suspicious function names.

In Section 3.8, we have shown that a fingerprinting site is more likely to use a variety of fingerprinting methods than it is to use just one. Thus, the more usage of suspicious APIs we detect, the higher the likelihood that we have found a fingerprinter.

A site could be labeled more suspicious if many fingerprintable APIs are used within the same script, and less suspicious if their use is spread across different scripts. However, a site looking to avoid our detection mechanism could simply spread their fingerprinting algorithm across multiple JavaScript files. In addition to this, many modern websites use *webpack*[1] or a similar technology to combine multiple JavaScript files into one when serving them to their users. This could lead a non-fingerprinting site to be labeled more likely to fingerprint than it actually is.

4.1 Algorithm

As mentioned in Chapter 3, PrivacyScore uses one of six distinct values to rate a characteristic of a site. Our algorithm shall therefore produce a rating on a six-value scale. The higher this score, the more confident we are that a website employs fingerprinting. As discussed in Section 3.10, we have listed certain giveaways, like the magic strings that are exclusively used in fingerprinting. Should we find any of these, we can immediately set the score to its highest possible value. Otherwise, we must increment the score for every fingerprintable API we find. We will give a perfect score of zero, which indicates no fingerprinting, only if a site uses no fingerprintable attributes or APIs. This is a rare case, as many modern sites use at least the *languages* attribute to set the user's language, or the *userAgent* string to perform certain user-experience enhancing optimizations. We do this because, as the study by Friedrich-Alexander-University (cf. Sec. 2.1.5) has shown, even just the *userAgent* can be unique among a large set of users.

We present the algorithm as pseudo code in Figure 4.1. The Canvas API must not only simply be used in any way to contribute to the scoring, but rather one of *toDataURL* or *getImageData* (or both) need to be used for it to be counted. As we have shown the Audio

1. https://webpack.js.org/

```python
def compute_score(results):
    score = 0

    # API use
    api_count = 0

    for api in APIs:
        if api in results:
            api_count += 1

            if api in ["canvas", "audio"]:
                score += 1  # extra score

    score += int(api_count / 2)

    # font use
    if 'mmmmmmmmmmlli' or 'Cwm fjordbak ...' in results.magic_strings:
        score += 5

    if results.fonts_set > 5 and results.measure_text_calls > 0:
        score += 1

    # known fingerprinting scripts
    for script in known_scripts:
        if script in results.scripts:
            score += 5

    # suspicious function names
    for name in suspicious_names:
        if name in results.names:
            score += 1

    # basic attributes
    if results.basic_attributes_count > 0:
        score += 1

    return score
```

Figure 4.1: Our scoring algorithm in Python-like pseudo code.

Table 4.1: Scoring results for fingerprinters

score	**number of sites**
0	0
1	1
2	1
3	4
4	5
5	19
total	30

Table 4.2: Scoring results for non-fingerprinters

score	**number of sites**
0	0
1	14
2	4
3	12
4	0
5	0
total	30

and Canvas APIs to be especially suspicious when it comes to fingerprinting, their presence gives an extra score point.

A score of zero corresponds to the PrivacyScore rating of "doubleplusgood", suggesting no fingerprintable APIs or attributes are used. A score of five means a "critical" rating, suggesting the use of many fingerprintable attributes or APIs, or the use of a known fingerprinting script or magic string. A score larger than five is reset to be exactly five, as this is already the maximum rating. A scoring of all 4929 sites in our dataset gives us an arithmetic mean of about 1.97, and a standard deviation of about 1.55, suggesting that the average site is scored "neutral", but that there is a fairly high variability of scores.

To evaluate our algorithm, we score 30 sites which we have manually identified as fingerprinters, and 30 sites which we have identified as non-fingerprinters. These sites were picked at random from our dataset. The results are shown in Table 4.1 for the fingerprinting sites, and Table 4.2 for the non-fingerprinting sites.

Table 4.1 shows that most fingerprinting sites get a score of three or above, with a heavy bias for a score of five. There are two outliers with scores of one and two respectively. The site with the score of two is kp.ru, which uses just the Audio API to fingerprint, and is thus an anomaly in our dataset. The site that was assigned a score of one is zoomit.ir, which uses only the WebRTC API, and is thus another anomaly. We base this on our findings presented

Table 4.3: Second scoring results for fingerprinters

score	number of sites
0	0
1	0
2	0
3	5
4	3
5	22
total	30

Table 4.4: Second scoring results for non-fingerprinters

score	number of sites
0	0
1	28
2	2
3	0
4	0
5	0
total	30

in Section 3.9, where we showed that the Audio and WebRTC APIs are the least used in our dataset. If we map the scores to the rating system of PrivacyScore, most fingerprinting sites would have been scored with a rating of "bad" or "critical".

Table 4.2 shows that most non-fingerprinting sites have a score of three or below. No site was given a score of four or five, but a score of zero is also absent. This is expected, as we only give a score of zero if absolutely no fingerprintable attributes or APIs are used. If we map the scores to the rating system of PrivayScore, most non-fingerprinting sites would have been scored with a rating of "neutral" or "good", but a significant portion would have been scored with a "warning" rating.

We conclude that fingerprinting sites will generally be given a score of three or more, and non-fingerprinting sites will generally be given scores of three or below. Give that a score of three maps to the rating "warning", and that most fingerprinters score above three, we contend that this is acceptable.

To consolidate this conclusion, we have scored 30 different fingerprinters as well as 30 different non-fingerprinters. The results are shown in Table 4.3 for the fingerprinters and Table 4.4 for the non-fingerprinters.

The fingerprinters in this set show roughly the same distribution as with the previous set. There are no scores below three, and most scores are a five. The non-fingerprinters show

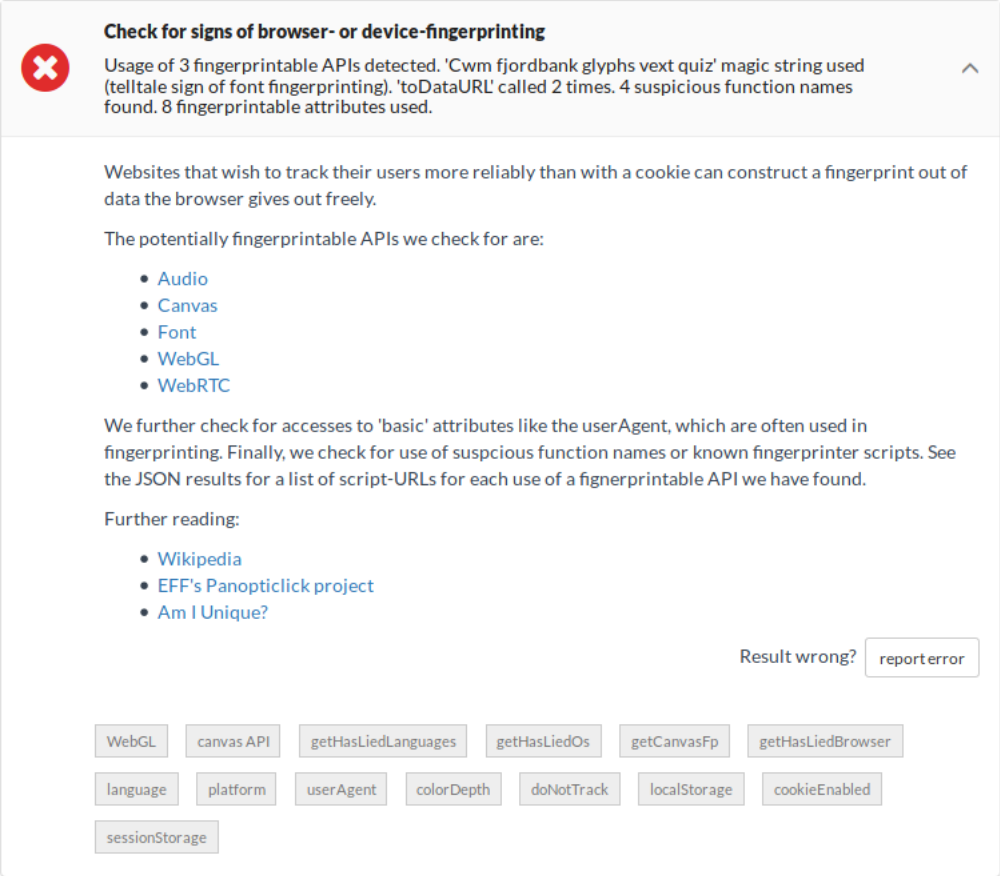

Figure 4.2: Screenshot of a fingerprinting result in a PrivacyScore results page. This particular result was generated for http://reddit.com.

even better scores than the previous set, with almost all scores at one, two scores at two, and no other scores higher than that. We conclude that our algorithm should perform well, and produce few or no false positives or negatives.

4.2 Implementation Details

The various tests that PrivacyScore performs are structured into different *test_suites*. As our algorithm relies entirely on data collected by OpenWPM, we integrate our test into the already existant *openwpm* test suite. We add a module in the *test_suites* directory which can analyse an OpenWPM database and produce the required data to be used by our algorithm, add an import to the *openwpm* test suite, and a call to our newly added module. A screenshot of fingerprinting-related results that can now be seen in PrivacyScore is shown in Figure 4.2. Physical distributions of this thesis will include the patched version of PrivacyScore on the attached USB drive.

5 Conclusion and Outlook

In this chapter, we summarize the findings and accomplishments of the thesis, as well as its limitations and shortcomings. We will also suggest topics for future work.

5.1 Summary

Many websites on the internet today wish to track their visitors. Setting a cookie is one option these sites have. However, a user can delete a cookie, and EU law even requires websites to display a notice when they use cookies. A browser- or device-fingerprint can be generated instead. Using the right attributs and APIs, it can contain enough entropy to uniquely identify a single user among a set of hundreds of thousands.

We have shown that the Audio, Canvas, Font, WebGL, and WebRTC APIs are all used for fingerprinting, and oftentimes, more than one API is used to construct a fingerprint. In addition to these APIs, basic attributes of the browser, such as the *userAgent* string, are used to enhance a fingerprint.

We have implemented an algorithm, which we have integrated into PrivacyScore. This algorithm is able to reliably identify websites that exhibit fingerprinting behavior. We have tested this algorithm on a set of 120 sites, and it is backed by analyses we have performed on random samples of a large, 4929 site dataset.

We have not explored APIs that have not already been mentioned in related work, and thus our implementation may miss fingerprinters that use such an API. Non-canvas based font fingerprinting can potentially be detected by our implementation via the *offsetWidth* and *offsetHeight* properties, but we have shown this to be not practical. More detailed WebGL instrumentation was not available when we collected our dataset, and we have performed only a limited analysis on the *WebGLRenderingContext*, using instrumentation we have patched into OpenWPM.

5.2 Future Work

5.2.1 Font Fingerprinting

Our implementation can detect canvas based font fingerprinting. We have shown that we can potentially also detect non-canvas based fingerprinting, though we have deemed our approach impractical. We leave better detection of font enumeration with the goal of creating a fingerprint to future work.

5.2.2 WebGL

As WebGL is not instrumented by OpenWPM at this time, we have patched instrumentation of the *WebGLRenderingContext* into the OpenWPM version used by our instance of PrivacyScore. We have not performed detailed analyses of WebGL fingerprinting mechanics, and leave this to be done in future work.

5.2.3 Fingerprinting Defense

This thesis has looked exclusively at fingerprinting detection. We have not explored potential defenses against it. The obvious starting point for fingerprinting defense are web browser. Firefox, for example, already takes certain precautions against fingerprinting, like with the Gamepad API.[1] However, modern browsers still expose a lot of fingerprintable information. Some APIs are even used primarily for fingerprinting or other malicious behavior. For example, we have found only a single legitimate use of the Audio API. We leave considerations on possible changes to browser behavior for future work.

1. https://developer.mozilla.org/en-US/docs/Web/API/Gamepad_API/Using_the_Gamepad_API

A OpenWPM Browser Params

The following configuration for OpenWPM's *default_browser_params* was used.

```
{
    "extension_enabled": true,
    "cookie_instrument": true,
    "js_instrument": true,
    "cp_instrument": true,
    "http_instrument": true,
    "save_javascript": false,
    "save_all_content": false,

    "random_attributes": false,
    "bot_mitigation": false,
    "disable_flash": true,
    "profile_tar": null,
    "profile_archive_dir": null,
    "headless": true,
    "browser": "firefox",
    "prefs": {},

    "tp_cookies": "always",
    "donottrack": false,
    "disconnect": false,
    "ghostery": false,
    "https-everywhere": false,
    "adblock-plus": false,
    "ublock-origin": false,
    "tracking-protection": false
}
```


Figure A.1: The Browser Parameters used by OpenWPM in our Crawls

B Canvas Fingerprinting Example

The code of Figure B.1 displays fingerprinting code using the Canvas API. It was taken from Fingerprintjs2 (cf. Sec. 2.1.3). We have made minimal modifications by removing comments for the sake of brevity, and changing long lines to make the code fit onto the page. The code is otherwise shown as-is, with not modifications that alter its functionality.

```
getCanvasFp: function() {
  var result = [];
  var canvas = document.createElement("canvas");
  canvas.width = 2000;
  canvas.height = 200;
  canvas.style.display = "inline";
  var ctx = canvas.getContext("2d");
  ctx.rect(0, 0, 10, 10);
  ctx.rect(2, 2, 6, 6);
  result.push(
    "canvas winding:"
    + ((ctx.isPointInPath(5, 5, "evenodd") === false) ? "yes" : "no")
  );

  ctx.textBaseline = "alphabetic";
  ctx.fillStyle = "#f60";
  ctx.fillRect(125, 1, 62, 20);
  ctx.fillStyle = "#069";
  if(this.options.dontUseFakeFontInCanvas) {
    ctx.font = "11pt Arial";
  } else {
    ctx.font = "11pt no-real-font-123";
  }
  ctx.fillText("Cwm fjordbank glyphs vext quiz, \ud83d\ude03", 2, 15);
  ctx.fillStyle = "rgba(102, 204, 0, 0.2)";
  ctx.font = "18pt Arial";
  ctx.fillText("Cwm fjordbank glyphs vext quiz, \ud83d\ude03", 4, 45);

  ctx.globalCompositeOperation = "multiply";
  ctx.fillStyle = "rgb(255,0,255)";
  ctx.beginPath();
  ctx.arc(50, 50, 50, 0, Math.PI * 2, true);
  ctx.closePath();
  ctx.fill();
  ctx.fillStyle = "rgb(0,255,255)";
  ctx.beginPath();
  ctx.arc(100, 50, 50, 0, Math.PI * 2, true);
  ctx.closePath();
  ctx.fill();
  ctx.fillStyle = "rgb(255,255,0)";
  ctx.beginPath();
  ctx.arc(75, 100, 50, 0, Math.PI * 2, true);
  ctx.closePath();
  ctx.fill();
  ctx.fillStyle = "rgb(255,0,255)";
  ctx.arc(75, 75, 75, 0, Math.PI * 2, true);
  ctx.arc(75, 75, 25, 0, Math.PI * 2, true);
  ctx.fill("evenodd");

  result.push("canvas fp:" + canvas.toDataURL());
  return result.join("~");
},
```

Figure B.1: Example of Canvas Fingerprinting, taken from Fingerprintjs2

C Font Fingerprinting Example

The code in Figure C.1 was taken from a script served by bankofamerica.com. It is not a complete script, but rather just one function definition in a larger script. We have "beautified" the code using DuckDuckGo's JavaScript beautifier, written by Akansh Gulati[1]. We have also modified the beautified code further to eliminate long lines, thus allowing it to be displayed on the page. We have not made any functional modifications.

The code demonstrates the use of the *mmmmmmmmmmlli* magic string in a *span* HTML element. It is used for font fingerprinting without the use of a Canvas element. The original URL of the full JavaScript was https://secure.bankofamerica.com/login/sign-in/cc.go.

In the code, multiple *span* elements are created and inserted into the DOM, each containing the magic string with a font size of 72 pixels, and with their font-family set to different combinations of fonts and the font types "monospace", "sans-serif" and "serif". These elements are invisible from the user. Their *offsetWidth* and *offsetHeight* properties are read, compared, and used for fingerprinting. These properties can reveal whether a font is available on the system, because if the font is not available, they will be the same as for the default, fallback font. So, if they differ for a certain font, that means the font is installed. If they do not, the font is not installed. The list of fonts used in the script contains 300 entries, but is not shown for the sake of brevity.

1. https://github.com/akanshgulati

```
tb = function() {
    var a = ["monospace", "sans-serif", "serif"],
        b = document.getElementsByTagName("body")[0],
        c = document.createElement("div");
    c.setAttribute(
        "style",
        "visibility: hidden;position: absolute; top: 0px; left: -999px;"
    );
    b.appendChild(c);
    b = document.createElement("span");
    b.style.fontSize = "72px";
    b.innerHTML = "mmmmmmmmmmlli";
    var d = {},
        e = {},
        f;
    for (f in a) {
        b.style.fontFamily = a[f];
        c.appendChild(b);
        d[a[f]] = b.offsetWidth;
        e[a[f]] = b.offsetHeight;
        c.removeChild(b);
    }
    this.detect = function(b) {
        var f = document.createElement("div");
        f.setAttribute(
            "style",
            "visibility: hidden;position: absolute; top: 0px; left: -999px;"
        );
        for (var g = [], k = [], h = 0; h < b.length; h++) {
            var l = [];
            k.push(!1);
            for (var n in a) {
                var p = document.createElement("div"),
                    q = document.createElement("span");
                q.style.fontSize = "72px";
                q.innerHTML = "mmmmmmmmmmlli";
                q.style.fontFamily = b[h] + "," + a[n];
                p.appendChild(q);
                f.appendChild(p);
                l.push(q)
            }
            g.push(l)
        }
        c.appendChild(f);
        for (h = 0; h < b.length; h++)
            for (n in l = g[h], a) {
                q = l[n];
                p = q.offsetWidth != d[a[n]] || q.offsetHeight != e[a[n]];
                k[h] = k[h] || p;
            }
        c.removeChild(f);
        return k
    }
},
```

Figure C.1: Example of *mmmmmmmmmmlli* magic string use by bankofamerica.com

Bibliography

[1] Gunes Acar et al. "FPDetective: dusting the web for fingerprinters". In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM, 2013, pp. 1129–1140. ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516674. URL: http://doi.acm.org/10.1145/2508859.2516674.

[2] Gunes Acar et al. "The web never forgets: Persistent tracking mechanisms in the wild". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2014, pp. 674–689.

[3] *Am I Unique?* URL: https://amiunique.org/ (visited on July 4, 2017).

[4] Peter Bright. *Adobe ending Flash support at the end of 2020*. URL: https://arstechnica.com/information-technology/2017/07/with-html5-webgl-javascript-ascendant-adobe-to-cease-flash-dev-at-end-of-2020/ (visited on March 18, 2018).

[5] Chair for Computer Science 1 of Friedrich-Alexander-University Erlangen-Nürnberg (FAU). *Study on Browser Fingerprinting*. URL: https://browser-fingerprint.cs.fau.de/ (visited on May 5, 2018).

[6] World Wide Web Consortium. *The canvas element*. URL: https://www.w3.org/TR/2011/WD-html5-20110525/the-canvas-element.html (visited on March 15, 2018).

[7] Peter Eckersley. "How unique is your web browser?" In: *Privacy Enhancing Technologies*. Vol. 6205. Springer. 2010, pp. 1–18.

[8] 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*. Ed. by Edgar R. Weippl et al. ACM, 2016, pp. 1388–1401. ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978313. URL: http://doi.acm.org/10.1145/2976749.2978313.

[9] Steven Englehardt and Arvind Narayanan. "Online tracking: A 1-million-site measurement and analysis". In: *Proceedings of ACM CCS 2016*. 2016.

[10] Amin FaizKhademi. "Browser Fingerprinting: Analysis, Detection, and Prevention at Runtime". PhD thesis. 2014.

[11] Electronic Frontier Foundation. *Panopticlick*. URL: https://panopticlick.eff.org/ (visited on July 4, 2017).

[12] CuteSoft Components Inc. *Javascript Obfuscator*. URL: http://javascriptobfuscator.com/ (visited on May 8, 2018).

[13] Martin Kleppe. *JSFuck - Write any JavaScript with 6 Characters: [](!+*. URL: https://jsfuck.com (visited on May 8, 2018).

[14] Pierre Laperdrix. *FP Central*. URL: https://fpcentral.tbb.torproject.org/ (visited on May 8, 2018).

[15] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. "Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints". In: *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE. 2016, pp. 878–894.

[16] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. "Mitigating browser fingerprint tracking: multi-level reconfiguration and diversification". In: *Proceedings of*

the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. IEEE Press. 2015, pp. 98–108.

[17] Max Maaß and Dominik Herrmann. "PrivacyScore: Improving Privacy and Security via Crowd-Sourced Benchmarks of Websites". In: *CoRR* abs/1705.05139 (2017). arXiv: 1705.05139. URL: http://arxiv.org/abs/1705.05139.

[18] Keaton Mowery and Hovav Shacham. "Pixel perfect: Fingerprinting canvas in HTML5". In: *Proceedings of W2SP* (2012), pp. 1–12.

[19] Nick Nikiforakis et al. "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting". In: *Security and privacy (SP), 2013 IEEE symposium on*. IEEE. 2013, pp. 541–555.

[20] Mike Perry et al. *The Design and Implementation of the Tor Browser [DRAFT]*. URL: https://www.torproject.org/projects/torbrowser/design/ (visited on February 8, 2018).

[21] Valentin Vasilyev. *Fingerprintjs2*. URL: https://github.com/Valve/fingerprintjs2 (visited on February 4, 2018).

Ich bin damit einverstanden, dass meine Arbeit in den Bestand der Bibliothek eingestellt wird.

Ort, Datum	Unterschrift

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Studiengang Bachelor of Science Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel — insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen — benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum	Unterschrift