



**UHASSELT**



**Maastricht University**

KNOWLEDGE IN ACTION

## **Faculteit Wetenschappen** **School voor Informatietechnologie**

master in de informatica

### **Masterthesis**

***A survey and evaluation of browser fingerprinting techniques***

**Ward Segers**

Scriptie ingediend tot het behalen van de graad van master in de informatica

### **PROMOTOR :**

Prof. dr. Peter QUAX

### **COPROMOTOR :**

Prof. dr. Wim LAMOTTE

### **BEGELEIDER :**

De heer Mariano DI MARTINO

De transnationale Universiteit Limburg is een uniek samenwerkingsverband van twee universiteiten in twee landen: de Universiteit Hasselt en Maastricht University.



**UHASSELT**

KNOWLEDGE IN ACTION

**www.uhasselt.be**

Universiteit Hasselt  
Campus Hasselt:  
Martelarenlaan 42 | 3500 Hasselt  
Campus Diepenbeek:  
Agoralaan Gebouw D | 3590 Diepenbeek

**2020**  
**2021**



**Maastricht University**

# **Faculteit Wetenschappen** ***School voor Informatietechnologie***

master in de informatica

## ***Masterthesis***

### ***A survey and evaluation of browser fingerprinting techniques***

**Ward Segers**

Scriptie ingediend tot het behalen van de graad van master in de informatica

#### **PROMOTOR :**

Prof. dr. Peter QUAX

#### **BEGELEIDER :**

De heer Mariano DI MARTINO

#### **COPROMOTOR :**

Prof. dr. Wim LAMOTTE





Hasselt University  
Transnational University Limburg

Master Thesis

---

# **A survey and evaluation of browser fingerprinting techniques**

---

*Author:*  
**Ward Segers**

*Assistant:*  
Mariano Di Martino

*Department*  
School for Information  
Technology

*Promotors:*  
Prof. Dr. Peter Quax  
Prof. Dr. Wim Lamotte

*A thesis submitted in fulfillment of the requirements  
for the degree of Master of Science*

June 2021



*“The Internet, my fickle friend, my two-faced enemy, what would life be like without you? Where else can I be anonymously anyone and yet, have no anonymity at all?”*

Susan Schussler



## *Acknowledgements*

First and foremost, my gratitude goes towards Mariano Di Martino, the assistant of my promoters, who was my supervisor. He spent hours on guiding me towards this thesis, keeping track of the progress, steering me in the correct direction, and proof-reading the thesis. This thesis wouldn't have been possible without him.

Next, I want to thank my promoters, prof. Peter Quax and prof. Wim Lamotte for their support and guidance. From the first conversation we had about my thesis proposal, the enthusiasm they had about the subject made me confident and motivated that it would be possible.

I would like all other education staff of Hasselt University for the five years in which I've learned basically everything I know about Computer Science. Without all of the knowledge that was passed onto me, it wouldn't be possible to write this thesis. Off course, all supporting staff of the university, who made this education possible, deserves an acknowledgement as well.

An acknowledgement goes to my fellow student Jeroen Bollen, who told me about the concept of browser fingerprinting near the beginning of my student career. From that moment onwards, the topic stayed in the back of my head as a possible subject for my thesis.

Finally, my go to all the members of my family and friends who supported me during the writing of this thesis. It should be said that writing most of the thesis fully remote with little interaction hasn't always been easy. It was nice to have people to fall back to.





# Contents

<b>Acknowledgements</b>	<b>v</b>
<b>1 Background</b>	<b>1</b>
1.1 Keeping state in the stateless	1
1.2 Transition to interactivity	2
1.3 Device fingerprinting	3
1.4 Incentives to fingerprint	4
1.4.1 Alternative to third Party Cookies	4
1.4.2 Captcha and bot detection	5
1.4.3 Fraud prevention	6
1.4.4 Discovering outdated software	7
1.5 Classification of fingerprinting techniques	7
1.5.1 Based on execution: Active ↔ Passive	7
1.5.2 Based on component	8
1.6 Fingerprinting other components	8
1.6.1 Server fingerprinting	8
1.6.2 Website fingerprinting	10
<b>2 Experiment setup</b>	<b>13</b>
2.1 Assessment criteria for fingerprinting vectors	13
2.2 Experiments: Fingerprint Lab	16
2.2.1 Global structure	17
2.2.2 Fingerprint library	17
2.2.3 Interface between Vue and the fingerprint library	18
2.2.4 Backend	18
2.2.5 Privacy concerns	19
2.2.6 Further improvements	19
2.2.7 Deployment	19
<b>3 Fingerprinting the browser</b>	<b>21</b>
3.1 Cache	21
3.1.1 Favicons	22
3.1.2 Cache Control	24
3.1.3 Alt-Svc	25
3.1.4 Mitigation	25
3.1.5 Conclusion	26
3.2 HTTP stack	26
3.2.1 HTTP versions	26
3.2.2 HTTP Headers	27
3.2.3 HTTP/2	27
3.2.4 HTTP/3	27
Transport-layer fingerprinting	28

0-RTT	28
Congestion control	29
Available drafts	29
3.2.5 Extensions	29
3.2.6 Plugins	30
3.3 NPAPI	31
3.4 Font probing	31
3.5 Conclusion	32
<b>4 Exploits using Web APIs</b>	<b>35</b>
4.1 Introduction	35
4.1.1 W3C Web Standards	35
4.1.2 Environment	35
4.1.3 GPU	36
4.2 HTML5 Canvas 2D font rendering	36
4.2.1 Experiment	37
4.3 Canvas 2D drawing without fonts	37
4.4 WebGL	38
4.4.1 Experiment	38
4.4.2 Mitigation	38
4.4.3 Conclusion	39
4.4.4 WebGL Debug Shaders	39
4.4.5 Experiment	40
4.4.6 Mitigation	40
4.4.7 Conclusion	40
4.5 WebGL Pixel buffers	40
4.5.1 Experiment	41
4.5.2 Mitigation	41
4.6 WebGPU	42
4.7 Web Audio API	43
4.8 Performance API	44
4.9 Battery	44
4.10 Screen	45
4.11 VR	45
4.12 WebXR	46
4.12.1 Experiment	46
4.12.2 Mitigation	46
4.12.3 Conclusion	46
4.13 Gamepad API	47
4.13.1 Experiment	47
4.13.2 Mitigation	48
4.13.3 Conclusion	48
4.14 Media Capabilities API	49
4.14.1 Experiment	49
4.14.2 Mitigation	50
4.14.3 Conclusion	50
4.15 WebRTC	50
4.16 Other components	51
4.16.1 Main memory	51
4.17 Conclusion	51

<b>5</b>	<b>Fingerprinting the Network</b>	<b>53</b>
5.1	IP address	53
5.2	Autonomous Systems	54
5.3	IP headers	55
5.4	IPv6-specific	55
5.5	TCP	56
5.6	Conclusion	56
<b>6</b>	<b>Fingerprinting prevention</b>	<b>59</b>
6.1	Incentives to block fingerprinting	59
6.2	Disabling JavaScript	60
6.3	Browser Built-in Solutions	60
6.3.1	Chromium-based	60
6.3.2	Firefox	61
6.3.3	Tor Browser	61
6.3.4	Safari	61
6.4	Differential Privacy and Privacy Budget	62
6.5	Conclusion	63
<b>7</b>	<b>Gaming as additional fingerprinting vector</b>	<b>65</b>
7.1	Selection of Techniques	65
7.2	Analysis	66
7.3	Further work	66
<b>8</b>	<b>Conclusion</b>	<b>67</b>
<b>A</b>	<b>Dutch summary - Nederlandse samenvatting</b>	<b>69</b>
A.1	Achtergrondkennis	69
A.2	Opzet van het experiment	70
A.3	Browser	71
A.4	Web API's	71
A.5	Network	73
A.6	Mitigatie	74
A.7	Gaming	74
A.8	Conclusie	75
	<b>Bibliography</b>	<b>77</b>



# Introduction

Browsers keep providing more and more functionality to developers. Functionality that was previously limited to natively ran applications. While these applications transform the Web into a platform, rather than a library of HTML files, it raises new privacy concerns, in the form of (a.o.) browser fingerprinting.

In this thesis, we conduct a domain study of the landscape of browser fingerprinting, a collection of techniques to uniquely identify clients, without relying on cookies, local storage or related technologies.

And, as fingerprinting vectors become evermore elaborate, so do protections built by browsers, extensions and other tools. We explore how browsers mitigate fingerprinting attempts and look at what balance can be found between privacy and functionality.

Our central questions are the following:

- Which fingerprinting vectors are relevant today?
- Which upcoming technologies provide a new attack surface for future fingerprinting?
- How can these attack vectors be mitigated, preferably with minimal impact on developers and end-users?

Also, this thesis aims to provide a theoretical base for game-based fingerprinting. In this technique, we use a game to strengthen our certainty about a fingerprint. Some central questions related to this are:

- Can we exploit certain game aspects to gain more data about the system and peripherals of a user's system?
- Can we use the availability of a game on a website to enable more fingerprinting? How significant would the potential increase in fingerprinting surface be?

Hence, we have two goals for this thesis. First, we try to provide an exploration of the existing techniques, ideally those which can be used in a browser fingerprinting game. Second, we will combine these techniques into a game which could be used to provide better fingerprinting.

## Structure

In the first chapter, we provide a brief history of browser fingerprinting, why it was developed and how people could protect themselves. We explain the important concepts and terminology that will be used throughout the thesis.

In the second chapter, a technical introduction is provided to the 'Fingerprint Lab', a framework that we developed to have a uniform way to implement and test potential

fingerprinting vectors. We discuss its technical setup and possibilities. We also lay out the parameters that we will use to assess techniques. These parameters allow us to compare fingerprinting methods better.

In the following chapters, we explore several categories of (potential) fingerprinting vectors, categorised by the component of a setup that they use. First, we look at browser fingerprinting. Next, we look at the various Web APIs that are available. Afterwards, we look at the network layer and the operating system. Network fingerprinting is a whole different research topic, but it can be used to strengthen the fingerprinting vectors, which is why we included it.

After that, we look at the general ways that users (or user agents) could take to avoid or combat this fingerprinting. While this will be discussed briefly after every attack vector, we choose to devote a separate chapter to general tips and approaches to the fingerprinting problem.

Next, we use this knowledge to discuss the ways at which a game could be used to fingerprint more effectively, compared to traditional methods.

Finally, we form our conclusion. We assess our work, discuss extensions to Fingerprint Lab, as well as further research that could be done in this regard.

*Dedicated to all family members and friends who supported me  
during these challenging times.*





## Chapter 1

# Background

To understand the problem of browser fingerprinting, it is important to understand the concepts that are its foundation. In this first chapter, we discuss what browser fingerprinting is, how it was made possible and what incentives exist to use its techniques. Also, we situate it in the bigger research field of fingerprinting.

### 1.1 Keeping state in the stateless

The original version of the HTTP protocol did not contain any form of state management. The original intent of the developers was to provide an open and collaborative World Wide Web for internal scientific uses. To that extend, the only state management required was authentication of users.

To facilitate authentication, the HTTP protocol contains the error code ‘401 Unauthorized’, which will ask user agents to provide credentials in order to continue [14]. In the original standard, these values were sent in plaintext [72]. In HTTP/1.1, hashing of the values was added [72].

As websites became more popular and complex, the need to preserve state between connections grew. RFC 2109 introduced the concept of cookies [43]. We can see cookies as key-value pairs, stored in the browser. When making subsequent requests to the website that issued the cookies, browsers should include the cookies as HTTP headers.

Cookies have several advantages. The first would be its relatively simple implementation. On client-side, all that is required is a database which links the domain with cookie keys and values. On server-side, depending on the needs of the cookies, no implementation might be required. If e.g. a language setting is given using a cookie, no server-side storage is required. Another advantage of cookies is their versatility. Any object that can be serialized into a string, can be stored in a cookie. A final advantage is the editability by clients, as user agents include debugging tools to allow modification of those cookies. Also, programmatic modification, such as through JavaScript, is possible.

Despite these advantages, several shortcomings exist. The first disadvantage is the fact that a user can control the contents of cookies from their browser. Users can decide to delete cookies, alter their contents or create them when no server has given them yet. When using these techniques to e.g. track which account a user is logged in on and which permission this user has, this can lead to security vulnerabilities.

Another disadvantage is the size limit imposed by cookies. As the values have to be stored in the browser of the user, cookies are limited to 4KB in size [43]. For some use cases, this will not suffice. While it would be possible to split items that would overflow this limit into multiple cookies, this solution could still give problems when browsers limit the amount of cookies that a website can store.

A third disadvantage is the lack of linkage to a particular setup. Criminals could try to steal cookies, insert them into their own browser and look at the website as if they were the original user. Attacks exist to extract cookies, such as cross-site scripting (XSS), malicious extensions and asking users to paste arbitrary code in the developer console. The original users could have no idea that their data is stolen and the server would have no idea that the requesting user agent is an imposter.

Some of these shortcomings, such as the editability, can be solved using session cookies instead of a raw value. These cookies store only an encrypted identifier in the cookie field, where related values are stored either on the server or in the cookie field in encrypted form. If a value were changed, the encryption would be broken (and thus invalidating the cookie). But, in offline-first applications, a server connection would be required to retrieve or validate the cookie.

A further improvement is the JSON Web Token (JWT), which consists of a JSON object that contains the relevant information (e.g. signed in user) and a signature to validate this data. The JSON is readable in the browser and the signature (made by a server) ensures integrity. However, in the case of linking the cookie to its setup, we've not mitigated the problem.

## 1.2 Transition to interactivity

As said before, the Web started as a collection of static resources. Since then, the Web slowly evolved to more of a platform, for which entire applications can be built. These applications can access functionality that would traditionally be reserved for native applications. One example of such application is videocalling. In order to function, it needs access to peripherals such as a webcam and microphone.

The concept of the web as a platform lead to Google creating a line of new devices, called Chromebooks. These devices rely on the ChromeOS operating system, which mainly consists of only the Chrome browser.

The first steps towards this interactivity were taken with the introduction of the Net-Scape Plugin API, shortened as NPAPI. Initially, the NPAPI, which was created in 1995 [50], allowed developers to create custom logic in webpages. API allowed developers to integrate their own frameworks into the browsers. We will discuss NPAPI more in depth in 3.3.

A popular NPAPI-framework to add interactive elements to static pages was the Flash framework. While it offered additional functionality (such as graphic acceleration and webcam access), it contained several vulnerabilities [12]. Also, users had to download extra software (for which they didn't always have the required administrator rights) and it was unavailable on Apple platforms due to the aforementioned security issues [50].

At the end of 2020, Adobe declared Flash End-Of-Life, after all mayor browsers had dropped support for NPAPI-plugins completely or are in the process of doing it [50].

Besides the security and privacy issues, the main reason for the discontinuation of NPAPI was the rise of HTML5 and related JavaScript-based APIs. These APIs can be found directly in the browser, with no additional software or third-party plugins required. Also, cross-platform compatibility is resolved by the browser, which will offer a standard set of APIs to the webpage. Also, because all logic remains in the browser process, security vulnerabilities overall had less impact than the NPAPI-based ones, where the plugin were to run in its own process.

As newer technologies emerge, new APIs are introduced to facilitate their usage on the web. A fairly new example can be found with Virtual Reality. As the technology became more popular, web developers wanted to be able to include the technology in their web pages. To that end, the WebVR proposed standard was developed [88]. While the standard is now deprecated in favor of the more generic WebXR standard, some browsers (such as Firefox) still support the feature [88].

## 1.3 Device fingerprinting

As we've seen, browsers now offer Web APIs to the webpages. However, not all browsers implement the same APIs, and some implementations differ, as we'll see later in this thesis. But, is it possible to deduct information from this? This is what the field of device fingerprinting tries to research.

Device fingerprinting is the concept to use information from several sources to identify a setup (with great probability). The name fingerprint is an analogy to the human fingerprint, which is unique to each human. As it is unique, the fingerprint could be used to identify people. But, oppose to e.g. a passport, changing a fingerprint isn't as easy. Therefore, using the fingerprint as additional measure of authentication is possible. A device fingerprint concerns collecting data from different properties to create a similar kind of fingerprint.

An individual technique to gain information is called a *fingerprint attack vector*, or shortened to *fingerprinting vectors* or *attack vectors*<sup>1</sup>. When talking about device fingerprinting in the context of browsers, the term *browser fingerprinting* is used. Browser fingerprinting is the subject of our research.

When done correctly, browser fingerprinting can be used in the same form of a session cookie, where it can be used as an identifier for the client, as can a human fingerprint. The biggest difference between the session cookie and browser fingerprinting is that users cannot delete or modify a fingerprint in the same way they could change a cookie.

To continue the resemblance with human fingerprints, it is worth noting that it can be difficult to know whether a party has collected information on a fingerprint. As some browser fingerprinting is done on the network layer, users don't know whether they are being fingerprinted.

The goal of the research domain is to get algorithms which can determine with great possibility how much certainty we have that a given connecting user agent is the same as one we saw before, potentially on a different website.

---

<sup>1</sup>Attack vector is often used in the context of ways to hack a system, but for the purposes of this thesis, we omit this meaning.

## 1.4 Incentives to fingerprint

Now that we have a basic idea about how browser fingerprinting works, we can wonder why companies and organizations would be eager to implement such features. As cookies provide authentication in a way that is easier to implement and manage, and offers more stability (fingerprints might differ slightly thanks to changes on the user's side), there seems little reason to do so.

In this section, we look into over several popular reasons to fingerprint users.

### 1.4.1 Alternative to third Party Cookies

In the recent years, massive amounts of data were stolen and subsequently leaked [66]. Recently, Facebook had data from about half a billion users leaked online. Consumers get more aware of the importance of their privacy, which puts a strain on companies that rely on their ability to accumulate this data.

Starting in 2019, Firefox disabled third party cookies as a way to follow users' surfing behavior among the web [89]. They did this by keeping cookies separated per domain that was visited, even third party cookies [89]. While this is beneficial for privacy, it can break previously working applications, such as single-sign-on (SSO), which relies on one set of credentials for multiple accounts.

While the market share of Firefox is a mere 3.7% [19], a similar decision has recently been announced by Google for their Chrome browser [64]. As this browser has a 64% market share (excluding all Chromium-based browsers) [19], advertising companies are now looking for new ways to track users online.

To avoid a situation in which every advertising company will be working on its own technique, the W3C has recently formed the Web Advertising group, which will work on a future Web Standard to improve both the advertising ecosystem and the privacy controls for users [63]. But, as of now, this standard isn't even in proper development yet. Therefore, a lot of uncertainty about the direction of online advertisement remains [63].

Another initiative that gained some tension, is the UID 2.0 initiative, which would simply ask users for an email address or phone number before accessing content [77] [22]. As this method would directly link data to users by their mobile phone number, instead of browser clients, the method was considered more intrusive than the previously disabled third party cookies. Google decided not to back the initiative [62] [63].

Google has proposed their own idea that doesn't use browser fingerprinting, FLoC. This algorithm would analyze a users' browser history and place those users into categories with people who have a similar history. While this would be more private than cookies, the overall privacy concerns remain. All other major browser vendors have said not to support the concept [18].

At the time of writing, no new standard has been formed. Several proposals are on the table, with some unclarity about which one will be the new standard.

As the goal of browser fingerprinting is to track user agents regardless of cookies, some companies might consider to implement some fingerprinting vectors to have some ability of cross-domain tracking.



**Figure 1.1:** An example of an early Captcha

### 1.4.2 Captcha and bot detection

Not all browsers are actually facing a human. Headless browsers (user agents without actual rendering to a screen) are used in all sorts of applications. Some use cases are the rendering of a webpage to PDF, being able to process the contents (as e.g. used by citation generating websites) or to conduct automated tests on a website in development [29].

However, these browsers can be utilized for less desirable use cases. One such usecase is to post spam on websites or to brute force a signin. More recently, during the computer chip shortage, bots were built to scalp available components. For these cases, some form of verification needs to be implemented on websites that can check whether users are human [35].

The most widespread way to identify whether a visitor is an actual human, is the use of so-called Captchas<sup>2</sup>. These tests will provide some form of test to see whether the website is actually being visited by a human [68] [83].

In the initial days of the Internet, these tests were rather simple. Some examples are '4 + 5 =' or 'What text is in this image?' However, as the fields of Computer Vision and Artificial Intelligence expanded, so did computers' ability to solve these tests, meaning that they became unusable to tell computers apart from humans [83] [17].

Over the years, these tests became increasingly difficult. At first, text would get mangled or had a lot of noise in the image. One such example can be seen in Figure 1.1.

Currently, the most popular Captcha is the Google ReCaptcha system, which uses a combination of browser fingerprinting and actual tests [17]. To activate it, users tick a box with the text 'I am not a robot'. After that, the system determines whether an additional Turing Test is required. If the system is confident enough using only a combination of browser fingerprinting vectors, it will not ask for additional verification. At the time of writing, the extra test consists of recognizing items on a matrix of pictures. These items are related to road signals and things that can be seen from the roadside. This ensures that almost all members of a general audience should know these items and be able to recognize them. As a benefit to Google, users help to train Machine Learning models used in Google's technologies [58] [53].

As Machine Learning models will become better at recognizing items on pictures, the ReCaptcha system is giving users more noisy images with more difficult challenges. This ensures that challenges remain difficult for an 'average botnet', without becoming impossible for humans [58].

It should be noted that, while Google is the main provider of Captcha technologies, it has been their services that were abused to break previous versions of their ReCaptcha [68].

---

<sup>2</sup>Completely Automated Public Turing test to tell Computers and Humans Apart

Other Captcha systems exist. hCaptcha is similar to Google ReCaptcha in workings, but doesn't use any browser fingerprinting. It only consists of the test element. hCaptcha allows customers who need a Machine Learning model trained to buy 'captcha space'. For this, hCaptcha gives a small financial reward to the websites using it, as a compensation for their users helping to enhance the machine learning models [1]. However, recent work shows that its simple structure might make it easier to bypass [32].

Another popular provider is Arkose Labs. Their tests consist of little games. Some examples are 'rotate the image until the correct way is up', 'pick the maze that has an exit', 'pick the die on the picture that add up to  $x$ ', and so on. As the images are of little contrast (or changing colors and textures in the maze example), it becomes more difficult to build automated solvers.

Captchas have had problems in the past with accessibility, as they tend to expect that users can view their screen. To address these problems, an audio option is often given. However, as audio recognition became better, these sounds had to become more difficult for users to hear. So, for people who require the accessibility, the rollout of Captchas often limits their ability to use websites [68]. If techniques using browser fingerprinting can provide adequate certainty about whether a given user is a human, the need for these Captchas might reduce. Cloudflare uses browser fingerprinting to reduce the amount of users that need to fill in a Captcha [16].

### 1.4.3 Fraud prevention

Cookies are a simple to implement scheme for authentication of users. But, it is possible to steal them from users. When using attack vectors (such as XSS attacks, deceptive browser plugins or letting users paste code in the browser's console), it is possible for attackers to steal session cookies from user agents. When signing in with these cookies, web servers will assume that the user agent is the one that was authenticated before. This practice is called session hijacking. The specific practice is called *cookie stealing* [67].

As these exploits can arise from outside the website, such as in the case of the console, cookies cannot be trusted as is on websites that have to be sure that the user is who they claim to be. Banks, for example, would have to make sure that it is actually their client who is executing the transaction.

An simple approach to try to ensure that the user agent with the given cookie is the user agent we authenticated, we can store the IP address (or AS number of that IP). This way, if the IP changed, the user session would be invalidated. Unfortunately, this approach came with a few drawbacks. As more data is consumed on mobile networks in a moving environment (such as in cars), so would their IP address change rapidly. This gives users a bad experience if they check an account on both their home and office Internet connection using their phone. Also, if hackers could abuse a VPN (or e.g. a malicious browser plugin that re-routes their traffic through the victim's connection), the detection would not work. Specifically for the AS-approach, if the hacker were to be on the same network as our user, no detection would take place. The final disadvantage of linking IP addresses to cookies would be IPv6, which can have devices have a multitude of addresses, which can change more quickly.

The use of more advanced options than IP addresses would yield both better detection of fraud and a better user experience. This would require us to distinguish devices in a more fine-grained manner. This is where browser fingerprinting comes in handy. By linking other data, such as the user agent properties, to the cookie (and verify this on server side), we can actually provide more certainty that a particular cookie is in fact used in the browser it originated in.

When in doubt, websites can choose to either completely invalidate the cookies. This would be effective against hackers, but can leave some honest users in the dust (as some configurations might change fingerprints often or just work against fingerprinting). Therefore, a better approach would be to ask for an additional factor of authentication. This factor could be a token sent by SMS or TOTP-based application. This concept is known more widely as *Risk-Based Authentication* (RBA), where multiple factors of authentication can be asked when the operation is deemed of higher risk [60]. While this can be in connection with fingerprinting mismatches, it can be in function of the type of action performed (e.g. sending small amounts of money to known recipients or a large amount to an account that wasn't used before).

Protection against session hijacking can be executed on the client-side as well. SessionShield is a browser plugin that will check whether a script is accessing the cookies [48].

#### 1.4.4 Discovering outdated software

Lots of web applications these days are of great importance and require the entire environment to be safe. Banking applications allow for the transfer of big amounts of data. While financial institutions might spend time and effort into securing their servers, these are only one half of a transaction. Are clients keeping their systems secure?

A survey shows that, to this date, 21.66% of users browse on an unsupported version of Windows [19]. Microsoft doesn't provide fixes for security vulnerabilities and other updates. This means that they're more vulnerable to attacks.

Financial institutions could use browser fingerprinting to determine whether users run outdated software. If they did, they could use this data to inform their customers to upgrade software. They could even restrict the use of applications for outdated software versions to ensure that transactions remain secure.

## 1.5 Classification of fingerprinting techniques

There exist several ways to classify different fingerprinting techniques, depending on the criteria on which the division is made. We will discuss a few in this section.

### 1.5.1 Based on execution: Active ↔ Passive

A first classification would be into active and passive techniques.

Active attacks are attacks that actively query the system to some data [42].

As active attacks have to query the system, they can be detected by the system.



In our thesis, the WebGL vendor information (4.4) is an example of a fingerprinting vector that is considered active, as it relies on the outcome of the WebGL API call.

On the opposite, passive attacks are attacks that don't involve a query to the system [42]. They solely rely on the behavior with which the system interacts with the server [42].

Please note that this behavior by the system can be instructed by the website, meaning that it can still be detectable in some cases. Some fingerprinting vectors, such as network based ones, don't involve any extra interaction with the user agent. All fingerprinting is done on the server.

Cookies themselves can be considered a passive fingerprinting vector [42], as they require no additional query step to retrieve. Besides that, we can also use a variety of headers provided by the browser (see 3.2.2) as example of a passive fingerprinting approach.

Passive attacks are more difficult to detect and mitigate, as it cannot have certainty whether fingerprinting is taking place.

### 1.5.2 Based on component

The classification in 1.5.1 focuses on the way the attack vector is executed. We can also take a different classification and look at which component is being abused. In this section, we give an introduction to this classification, which we will use throughout the thesis.

The first category is the browser-specific components. Within this category, we include the application layer networking, as the application-layer networking is part of the browser from which it originates.

The second category is the available Web APIs. The third category is attack vectors based on lower-level networking, such as the transport layer (i.e. TCP) and the IP layer.

While we could consider the operating system as another category, we don't do that for the purpose of this thesis and include these techniques in the browser aspect.

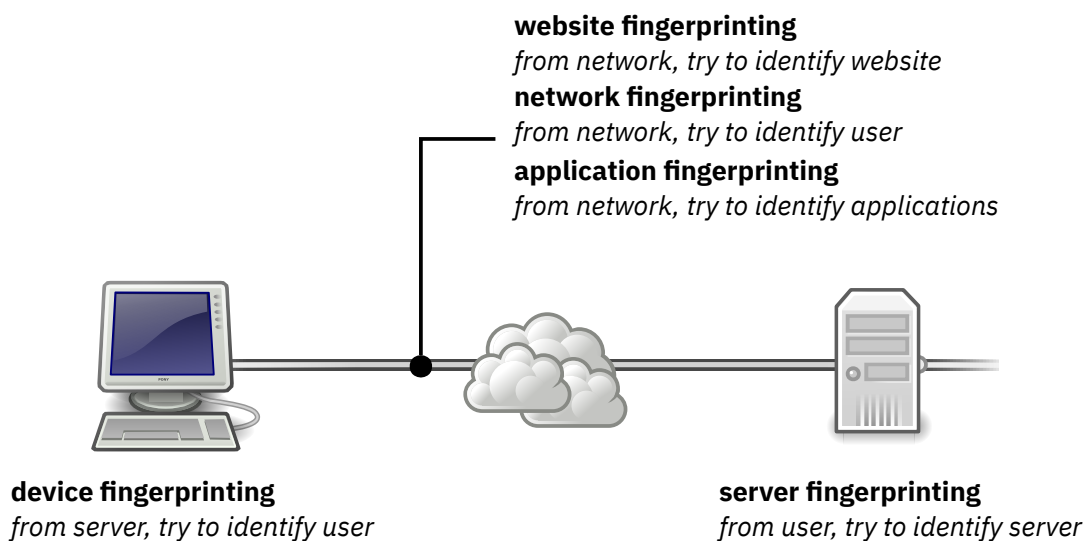
Another category can be found in the networking. Here, all other layers of the networking stack (transport to physical) can be considered. Not all layers can be fingerprinted by a website (e.g. the physical layer isn't, to our knowledge, impractical for websites to fingerprint).

## 1.6 Fingerprinting other components

Device fingerprinting isn't the only concept of fingerprinting. Many components of the Internet are fingerprintable. In this section, we show a brief summary of the 'fingerprinting landscape'. A visual representation can be seen in Figure 1.2.

### 1.6.1 Server fingerprinting

Server fingerprinting is a concept that uses techniques similar to device fingerprinting, but uses them to try and identify the server. It tries to uniquely identify which software is running on the server to which a user connects. This concept allows an



**Figure 1.2:** A visualization of the different elements of a connection to fingerprint. Browser fingerprinting, the subject of this thesis, is a subcategory of device fingerprinting

attacker to find outdated software, which can subsequently be used to find potential exploits.

The main difference is that websites can run arbitrary JavaScript code when the webpage is loaded. The concept of running code doesn't apply on webserver. This means that server fingerprinting is mostly related to either application layer or network layer data, using a passive approach.

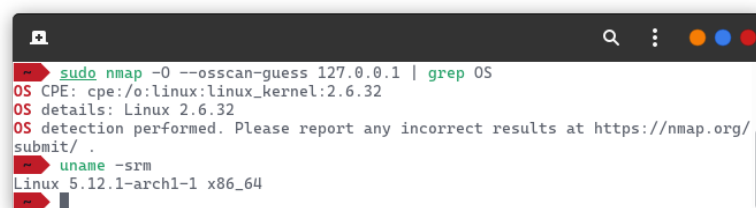
An important component to fingerprint is the software serving clients on ports 80 and 443, the webserver. Fingerprinting these webserver is called *web server fingerprinting*. The most commonly used webserver are Nginx and Apache HTTPD [79].

As we want to provide a general overview of server fingerprinting, we will provide some examples as to how to fingerprint a web server.

A first technique is to look at server-specific settings that can be seen as a visitor of the website. Among these values are the Server-header (similar to the User-Agent header on HTTP requests) and the default index and error pages [26]. To avoid exploitation of these vectors, website administrators can omit the header, delete the version number or replace its content with a custom value, e.g. a website specific value. Error pages can be replaced with website-specific pages as well. The practice of extracting the headers from a protocol to gain information about the system is called *banner grabbing* [6].

An example as to determine the operating system on which the webserver runs, is the use of TCP scanning. By using this technique, an application looks at TCP parameters and congestion control algorithms used. As it contains a database with which operating system versions default to which parameters, a guess can be made as to which operating system is running [26].

Automatic tools to exploit these techniques exist in tools such as nmap. As a small

A terminal window with a dark background and light text. The command `sudo nmap -O --osscan-guess 127.0.0.1 | grep OS` is entered. The output shows: `OS CPE: cpe:/o:linux:linux_kernel:2.6.32`, `OS details: Linux 2.6.32`, and a message about reporting incorrect results at <https://nmap.org/submit/>. Below this, the command `uname -srm` is entered, and the output is `Linux 5.12.1-arch1-1 x86_64`.

```
➤ sudo nmap -O --osscan-guess 127.0.0.1 | grep OS
OS CPE: cpe:/o:linux:linux_kernel:2.6.32
OS details: Linux 2.6.32
OS detection performed. Please report any incorrect results at https://nmap.org/submit/
➤ uname -srm
Linux 5.12.1-arch1-1 x86_64
```

**Figure 1.3:** Using nmap, we can identify our own laptop as a Linux machine, but the version number is incorrect. Other devices that we tested had similar or worse results.

experiment, we used it to try and identify our NAS, local DNS server and laptop (Figure 1.3). While it was able to correctly identify the laptop, it failed to identify a Synology NAS and Pi-Hole DNS server.

### 1.6.2 Website fingerprinting

Another fingerprinting component is website fingerprinting. Where browser fingerprinting tries to identify users from the perspective of a server, website fingerprinting is used to describe the process of identifying which website is being visited. This website fingerprinting is seen from the perspective of a network administrator, who has no control over either the webserver or client device [91]. Sometimes, the terms Website traffic fingerprinting or traffic fingerprinting are used to indicate that analysis has to happen on traffic.

As website fingerprinting is done per definition at the network layer, the techniques that are usable are a small subset of what client or server can do to fingerprint each other. As no code can be executed, special requests can be made or properties can be seen, no more. Also, as the attacker has (likely<sup>3</sup>) no knowledge over the contents of HTTPS traffic and cannot read which requests are made, although it is still possible to distinguish individual pages to a certain degree [23]. DNS analysis is another possibility, but newer standards such as DNS-over-TLS and DNS-over-HTTPS can make this approach more difficult in the future, as it would be more difficult to distinguish DNS queries from other HTTPS traffic.

An important incentive to identify individual websites being visited, is censorship. This can be because of government legislation requiring it, or because a network administrator doesn't want users to visit certain website from their network. One example would be the Great Firewall of China, a big 'filter' which blocks several western websites and content about subjects that the Chinese government doesn't want being talked about [21].

While not directly related, if website fingerprinting is done accurately and repeatedly on a network, individual user's browsing patterns could be derived from this technique, leading to the same ideas browser fingerprinting tries to achieve. However, it would be limited to individual IP addresses on the network on which the user is requesting the data.

---

<sup>3</sup>In the context of business networks, it is possible for network administrators to install a custom CA and subsequently read the encrypted traffic. For this explanation, we omit this exception.

We won't discuss website fingerprinting any further in this thesis, but will get back to it when discussing techniques that have similar ways of exploitation in the website fingerprinting field.



## Chapter 2

# Experiment setup

In this chapter, we explain the setup required to fulfill the experiment. First, we determine what parameters will be used to differentiate the different fingerprinting vector.

In the second part of this chapter, we elaborate on the framework that we developed to compare fingerprinting vectors.

### 2.1 Assessment criteria for fingerprinting vectors

Not all fingerprinting vectors are created equal. Some provide data can be used to identify a user on its own, while others only provide a very small detail of data. Therefore, we need criteria to assess the impact of different vectors to be able to compare them.

During this thesis, we will assess fingerprinting vectors using the following criteria:

- **Degree of entropy:** Some attack vectors will generate the same output for the majority of users, with only a very select amount of users yielding other results. While this means that we can fingerprint the users in the latter category with a bigger certainty, it does leave a big group without fingerprinting surface.

For this thesis, we refer to entropy as the degree of spread of fingerprinting results in the answer universe. When referring to the amount of data we can gain from a vector, we'll use the term *amount of data*.

Ideally, we would like every attack vector with  $n$  possible outcomes to have the same probability  $\frac{1}{n}$  to be selected, giving a maximal entropy. But, the combination of low entropy values with a select amount of high entropy values might be useful to identify individual users better.

Given a fingerprinting vector  $V$  with  $n$  possible outcomes, the entropy  $H(V)$  can be written as

$$H(V) = - \sum_{v=i-1}^n P(v_i) - \log_2 P(v_i) \quad (2.1)$$

with  $v_i$  being the probability of the  $i$ th element being retrieved as a result.

We understand that several fingerprinting vectors are related, but didn't account for this when calculating their entropy. For example, the appearance of the Safari browser is closely related to the appearance of the MacOS or iOS operating system, as these are the only operating systems where this browser is

available. We didn't take these into account, because we analyzed fingerprinting vectors in isolation, as the amount of combinations of rules that we should account for would become too big for our research. Also, we could (wrongly) assume two rules to be related or unrelated.

As we don't have a big userbase to test our experiments on, we will limit ourselves to analysis of statistical data that is available from various reputable sources.

- **Amount of extracted data:** Some attack vectors provide a binary value (e.g. the availability of a specific browser feature), while others can extract an entire image as entropy. We count the amount of extracted data in bits.

One problem we encountered with this counting technique, is that bits can rely on each others' value. One such example would be the WebGL attack vector, discussed in 4.5. As the WebGL API would be available, it wouldn't be possible to calculate a hash, meaning that the bits dedicated to the hash don't have to be accounted for. We allude to the amount of extracted data in the maximum amount of useful bits that we can extract, assuming all available features are exploited.

- **Stability of the result:** Fingerprint vectors might not always produce the same result, as the component on which the vector relies might be changed. We define the notion of stability as the amount of change required to a system to make the fingerprint vector produce another outcome, apart from a component providing actual mitigation.

The last part of our definition is important. To avoid having to account for 'the user installs a tool that actively blocks this vector' in every test where it is feasible, we don't count these forms of mitigation.

As the notion of stability is rather vague and we cannot find an easy formula for it, we create the following possible groups in which a vector can land.

1. Change in system state. We use this category to catch cases in which we cannot completely predict when the state of the fingerprint will change. Some examples are battery discharge and plugging out USB devices. It should be noted that changes in the internal hardware of the device aren't contained in this category, as are changes to the system settings.
2. Change in network. This indicates a change of network (e.g. a device that moved from one LAN to another). Changes in the version of the Internet Protocol also fall under this category.
3. Change in browser. This category includes changes to browser settings, clearing browser data, adding or removing add-ons etc. We exclude add-ons or settings that block the specific fingerprinting vector (for example, the Firefox setting 'resistfingerprinting' doesn't count).
4. Change of browser. This category indicates the change from one browser to another. Some fingerprinting vectors will work across browser engines, as e.g. Microsoft Edge, Google Chrome and Opera all use the Blink engine. So, if a change between these browsers doesn't give a difference in outcome, we will verify whether a browser with a different engine

(such as Mozilla Firefox or GNOME Web) does give another fingerprinting outcome.

5. Change in OS. This includes changes to the fonts that are installed, drivers that are available for different hardware devices, system settings etc.
6. Change in hardware. If the fingerprint relies on the actual properties of the hardware, the only way to stop the attack vector (apart from software solutions to block the access to the property) would be to physically replace hardware.

We don't include changes in the state of connected devices in this category. Those are contained in the first category. The reasoning behind this is that a change in hardware is rather uncommon (such as a desktop PC that gets a new CPU). Therefore, this category is more severe to mitigate.

If an outcome can be changed by multiple (e.g. both a browser and operating system change), we place it in the first one in which it appears, as we tried to order them to their likelihood of happening.

- **Impact of complete mitigation:** Some attack vectors are closely related to the core functionality of the component they exploit, where others provide specific functionality for a small userbase. In this category, we look at the impact of a complete mitigation of the vector.

To determine the impact, we created the following categories:

- Loss in performance. Some attacks rely on specific hardware to be available to accelerate calculations. Some hardware, such as GPUs, suffer from floating point errors, which can lead to stable fingerprints (see 4.5). These could be mitigated by computing those values on the CPU instead, where those can be avoided.

Running such computations on the CPU would yield significant performance loss, which would hurt the user experience if the website or application relies on performance.

- Loss in functionality. Certain vectors use elements of APIs which are closely related to the main functionality of that API. One such example is the availability of caching in browsers. The simplest and complete mitigation for this vector is to omit the cache entirely and re-download resources if a website is visited again. Bandwidth usage would drastically increase, hurting the functionality that was available in browsers.
- Loss in convenience. One way to block certain functionality from being exploited, is to ask users permissions for every component of the browser that the website tries to use. As asking permission to often leads to users clicking 'Yes' on all questions,

Multiple categories can co-exist for a certain vector, as multiple mitigations might exist that protect against the fingerprinting.

In some cases, such as the cache exploits discussed before, browsers change implementations of components to limit the usability of the attack in ways that



don't completely mitigate the fingerprinting vector, but e.g. make them unusable for tracking users cross-sites. Therefore, the impact of complete mitigation doesn't directly reflect the actual impact mitigations would have on average users.

- **Speed of execution:** Some fingerprinting vectors can produce a result immediately. Others require more time to return a value.

As exact measurements for this parameter are highly dependant on the hardware on which it runs, we use magnitudes of speed.

- Instant. The fingerprinting vector doesn't require any significant calculations to be done, or these calculations can be done on a server.
  - Fast. Some calculations have to be done. A small number of requests has to be made. Those requests can be done parallel.
  - Slow. A big number of calculations has to be done. A bigger number of requests have to be done.
- **Impact of execution:** Whereas the Impact of mitigation focuses on what were to happen if the fingerprint vector became unfeasible, the impact of execution focuses more on what the user would notice if this vector is executed.

This criterion is closely related to the speed of execution, as slow vectors would lead to users waiting longer for their website to load. Nevertheless, there are cases in which a fast execution has a big impact, and cases in which a slow execution has no impact.

An example for fast executed fingerprints with a big impact to users would be scripts that e.g. ask users to activate their webcam.

An example for a slow vector with no impact would be those vectors who use asynchronous calculations. These calculations (e.g. benchmarking) could be done in the background with little impact for the user.

- **Percentage of users vulnerable:** Some fingerprinting vectors only work on some configurations.

Here, we try to make an assessment of how many users would be impacted by this fingerprinting vector. As with degree of entropy, we only use statistics (and no field tests) to assess the severity of different attacks.

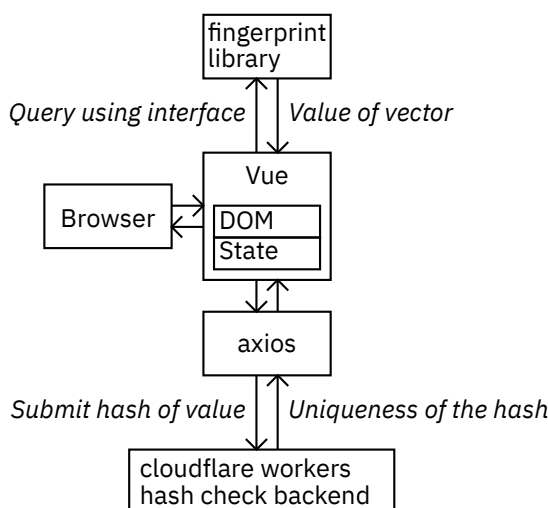
## 2.2 Experiments: Fingerprint Lab

When implementing proof-of-concepts for several fingerprinting vectors, we found that making several separate prototypes was inefficient. We were creating a very similar prototyping setup multiple times, with no easy option to compare the results that were obtained. It would give better opportunities to integrate these prototypes into a framework, where they could be used together. To this end, we developed a framework called 'Fingerprint Lab'.

The working of Fingerprint Lab is similar to the working of metasploit framework<sup>1</sup>. We allow users to pick which fingerprint vector(s) to use and execute them. For our research purposes, we also check whether we encountered a

In the future, we could expand the framework into supporting more fingerprinting vectors.

### 2.2.1 Global structure



**Figure 2.1:** The structure of Fingerprint Lab. In *italics* we see the data that is exchanged between components. Requests on the left are made, responses to them are on the right.

A global summary of the project can be seen in Figure 2.1. We will explain the different components and their communication in the following sections.

The project consist of a progressive web application and a RESTful API. The frontend uses the Vue frontend framework with the Vuetify components to quickly create a web application using Material Design. The backend consist of a TypeScript-written backend on Cloudflare Workers.

### 2.2.2 Fingerprint library

The fingerprint library is a collection of the implementation of different fingerprinting vectors. Besides the implementations, it also contains a short description of the fingerprinting vector and the values for the criteria discussed in 2.1.

Each fingerprinting vector has a unique identifier in the form of a string, to be able to include them into the URL of the pages that are being viewed. To subdivide the vectors, we put the vectors in categories.

<sup>1</sup>Metasploit is a framework with implementations for exploits. It allows all these exploits to be configured and run trough a uniform framework.

### 2.2.3 Interface between Vue and the fingerprint library

By using TypeScript, we created an interface<sup>2</sup> that we were able to implement for all fingerprinting vectors. Each fingerprinting vector in the fingerprint library adheres to the interface.

While we will not discuss every aspect of the interface, the main member is the actual handle to execute every fingerprint vector.

```
function execute(node: HTMLElement): Promise<T>
```

While some fingerprint vectors require access to the document and/or the window, this is no problem, as these are available as global variables. We did have a parameter `node`, as access to one particular element of the DOM might be preferred. The parameter was chosen to provide the possibility of several fingerprinting vectors to be ran simultaneously. If we e.g. used one id for the DOM element that the vector would require, we would not be able to run multiple vectors in parallel.

This approach makes our Fingerprint lab codebase ready to use in other environments (e.g. a production environment), where multiple fingerprinting vectors would be used at the same time.

The data collected changes per fingerprinting vector. To account for this, we made out interface return a generic type `T`, which is an object that can be shown in JSON format to the user. To make the data universally usable, each fingerprint vector contains a way to hash this value into a SHA512 sum.

The shortcoming with this is that some fingerprinting vectors don't require a node to be made. Our framework has currently no indicator to show this and will simply ignore the parameter if not required. Also, it might be better to allow multiple elements to be given to the function, instead of e.g. relying on the availability of a global document or navigator element.

### 2.2.4 Backend

The backend uses the Cloudflare Workers. The main reasons for this are its cheap offering and the availability of a Key-Value-Store. This KV-store gives us the possibility to keep track of fingerprints, without relying on a database. We store the fingerprinted data for 24 hours in the KV-store and delete it if no other client matched this value, which makes the framework more privacy-friendly.

As said in 2.2.3, supported fingerprint vectors could use the backend by sending the hash, generated out of output data of the fingerprint value. The backend will then search on the key for this combination of hash and vector. If it finds a hash collision, its value (the nicknames of configurations that gave identical results) is returned. Otherwise, the backend indicates that it is unique.

We used the KV-store's expiration feature to delete KV-pairs that weren't queried for over 24 hours.

---

<sup>2</sup>Interfaces in TypeScript work like in Java, where they provide a list of function signatures.

### 2.2.5 Privacy concerns

As we store fingerprinted values from the devices on which we run the experiments, some privacy issues arise.

The most prominent problem is that we essentially collect fingerprinting data. While we only send hashes, this doesn't mean that the data isn't recoverable. Despite our hash function being irreversible, it could still be possible to know which results for a given fingerprinting vector are 'possible'. In the case of e.g. the WebGL Vendor information, we know which models of GPUs exist. If we hash all these possibilities, we know in which buckets these would fall.

As this technique could lead to us knowing the hashes that are likely used by our clients, it is also possible for external parties to get and combine multiple hashes into one fingerprint and linking them together as mentioned above.

To make sure the privacy risk for users is kept to a minimum, we try to fulfill as much as possible on the client side and minimize our server implementation. If users don't want to rely on the backend, they can still use the implementations. The framework can operate without the online component, meaning users can choose not to send their hashes to the server and use the application completely independent of the server.

### 2.2.6 Further improvements

As the main use of Fingerprint Lab is quick prototyping and looking for fingerprint collisions, several things were not looked into further.

One disadvantage is the lack of proper database support (the KV-store is intended for caching). If we chose a platform that could support proper database support (such as Amazon Web Services), we could have had more data about the fingerprints. But, as our focus was to preserve users' privacy and not collect more data than necessary to verify uniqueness, we did not continue with this.

Also, there is no check on whether the given hash values of clients are actually valid. Clients can send any (valid) hash to our server.

These shortcomings, alongside the fact that no spam prevention was implemented, should make it clear that no statistical relevant data can be derived from this experiment.

Another shortcoming is that the hash function used is SHA512. It is a general-purpose hash function, which means that values that are similar don't hash to a value close to each other. It might be better to change this to a fuzzy hashing scheme, such as ssdeep.

Also, it now only allows for individual vectors to be executed at one time. It would be a useful if multiple vectors could be executed at the same time.

### 2.2.7 Deployment

To deploy our application for use on mobile devices, it would be convenient to have a live version working.

The frontend and backend use different git repositories, which makes it easier to setup different CI pipelines for them to automatically deploy.

The frontend is served to users using Netlify.

A live version can be found on <https://fplab.wardsegers.be>. Although not accessible for humans (only PUT-requests are allowed), the accompanying backend is located at [https://fplab\\_backend.editicalu.workers.dev](https://fplab_backend.editicalu.workers.dev). Source code is provided as attachment to this thesis or available on request.

## Chapter 3

# Fingerprinting the browser

The main objective for browser fingerprinting is to gain information about the environment in which the website is shown. The software component that interacts directly with the website, the browser, would be a logical place to start exploring possible vulnerabilities. Which exposed components are unique to individual browsers?

In this chapter, we will go through current and historical fingerprinting vectors that exploit browser functionality. To do this in a structured manner, we subdivide every attack vector as follows. We first give an introduction to the subject and its fingerprint potential. We have a section for every element that can be fingerprinted. Next, we introduce our experiment (if we conducted one). We describe its setup and findings. Next, we examine the potential mitigations, their advantages and possible disadvantages. We try not to simply omit all the functionality, as this would hurt the user experience. Finally, we form a conclusion about the severity of the fingerprinting vector.

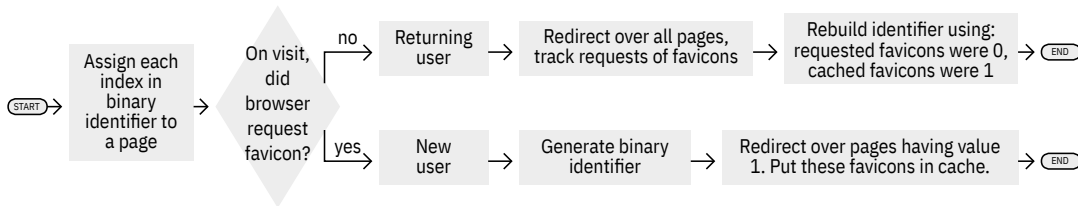
While we try to provide a comprehensive list of fingerprinting vectors, the field of browser fingerprinting is vast and still expanding. New technologies also lead to new vulnerabilities. As our focus is more towards the Web APIs in Chapter 4, this list is a selection of the most prominent (or historical) fingerprinting vectors.

This chapter mainly focuses on existing fingerprinting vectors. No new techniques were discovered in this chapter, but we do try to implement several known vectors into our Fingerprint Lab.

### 3.1 Cache

Browsers often include several forms of caching. The main reason to have a cache is to reduce bandwidth usage and provide users with a better experience. As users often visit the same sites, those resources can be retrieved from the server only once and kept in the cache. Since users 'browse' the web less often, and more often use a fixed set of websites [81], caching of a select set of often visited websites becomes more impactful.

HTTP has built-in mechanisms to assist the correct caching of website. On the first visit of a website (or the first one after a cleared cache), the webserver can choose to add the `Cache-Control` header, which indicates how long a file can be safely kept in cache. When revisiting the website, browsers can add a `If-Modified-Since` header to their HTTP request. In such case, the server can answer that the file hasn't changed (rather than retransmitting the page). In such cases, status code 304 `Not Modified` should be used.



**Figure 3.1:** A visualization the pipeline to assign an identifier to a viewer, or how to read the identifier of a returning visitor using the favicon fingerprinting exploit.

However, as this concept keeps data from sites visited before, it contains a lot of vulnerabilities around fingerprinting.

While we discuss the concept of ‘the cache’ in the following subsections, we need to point out that when talking about ‘the cache’, we mean the different caches available in browsers.

In the following subsection, we examine several known attack vectors considering the cache.

### 3.1.1 Favicons

During the writing of this thesis, several new exploits were found. One of these included the use of favicons [71].

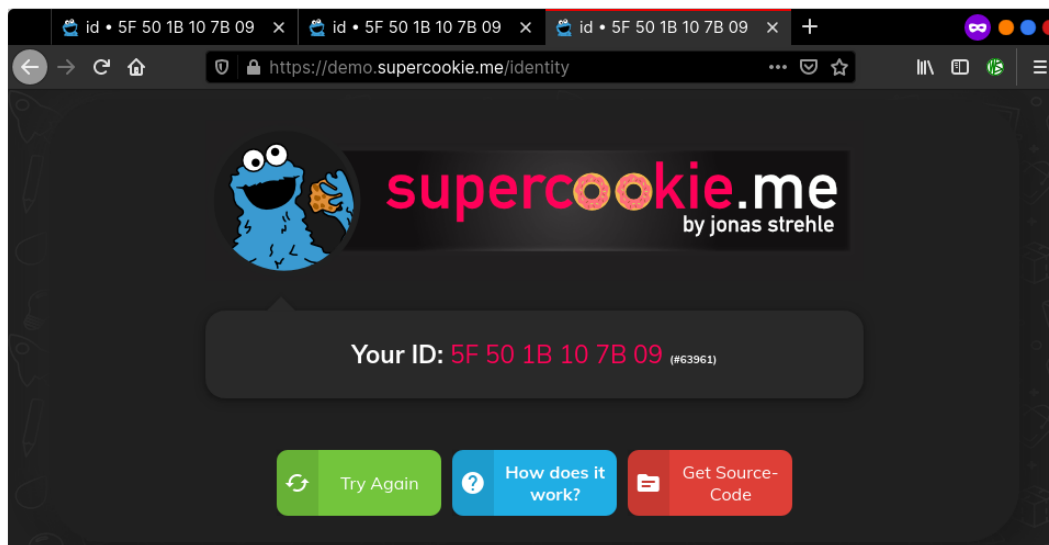
Favicons have historically been cached in a separate cache from the main browser cache. As their appearance is required for a correct display in the favorites bar. Before cache control existed for other elements of the browser, favicons had to be cached to accomodate this behavior.

The mechanism works as described in the following sections. The same principle can be seen visualized in Figure 3.1.

To find out whether a user visited the website before, we begin by checking whether the user requested the actual favicon of our website. If he did, the icon did not appear in the cache of the user and has thus not visited the website. When getting a request for a favicon, the server continues to ‘write’ the identifier. It does so by letting the client pass through a number of subpaths (using redirects), making the browser cache the accompanying favicons [71].

When the user didn’t request the original favicon, we can assume that it was in their cache, indicating that the user has visited our website before this day. Now, we send a request to every subdomain (in a redirect loop). Normally, all the favicons of subdomains that did were a 0-bit in our identifier would actually have to request the favicon, with the 1-bits not sending any such request. To avoid the problem that the 0-bits actually get stored in the cache, the server returns a 404 HTTP code, meaning that no value gets stored in the cache and the browser will try to request this icon again upon the next visit. This way, the attack vector can be used the next time [71].

The reason favicons specifically can be attacked, is twofold. On the one hand, it is a request that is traditionally done with any GET-request. The other reason is that favicons are historically cached separately from other resources, which lead to the problem that this separate cache wouldn’t be flushed if a user asked the browser to delete caches [71].



**Figure 3.2:** The fingerprint using the ‘Favicons’ exploit, explained in 3.1.1, retains its value over multiple runs (seen here as multiple tabs). The same fingerprinting doesn’t succeed outside of the incognito context.

As this vulnerability has been exposed recently, many browsers remain fingerprintable. We can’t create a proof-of-concept of this attack vector using our Fingerprint Lab framework, as it doesn’t support page redirects (because it would break the Progressive Web App). To conduct tests, we resided to an external implementation.

Our tests indicate that the latest release of Firefox (version 88) is patched and gives a new identifier every time we run the test. However, we see that the incognito window retains its fingerprint while opened, as can be seen in Figure 3.2. This is likely due to a different cache implementation for the incognito window, as its content will be removed as soon as the user closes the window. The same continues after we close the browser and open a new incognito window.

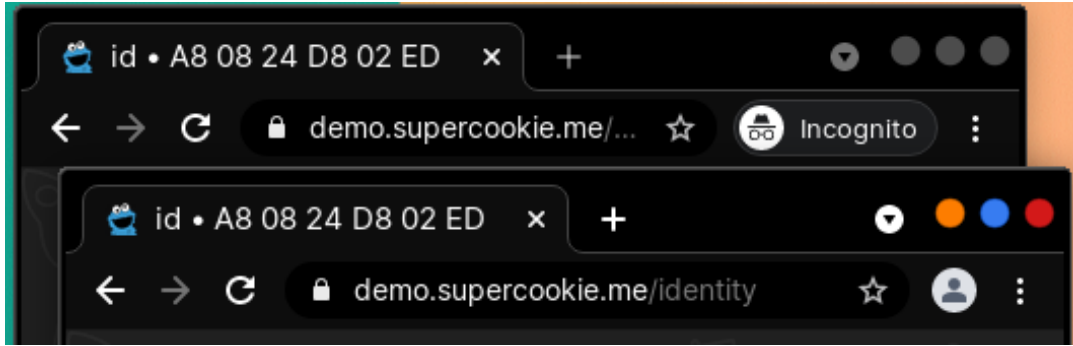
Chrome (version 90) is susceptible to the fingerprinting. Even worse here is that the fingerprint that Chrome generated, is persistent across the incognito window and the standard browser window, meaning that it could be used to track incognito sessions. This can be seen in Figure 3.3.

For a Webkit-based browser, we used GNOME Web. The browser is vulnerable to the same fingerprinting vector.

Another remarkable observation is that even the Tor Browser suffers from this fingerprinting method in some form. We conduct a test, using an existing proof of concept, and find that the most recent version of the Tor Browser (version 10.0.16) returns the same result in consecutive runs in a session. It is worth noting that, as this cache is cleared when the browser is closed, that this fingerprint cannot persist across sessions. Thus, the fingerprint vector cannot be used to track Tor users longtime, but it can be used to track their session.

The publication of the paper gave some controversy, as it wasn’t disclosed to developers who could create fixes for these fingerprinting vectors before publication and thus left users vulnerable [71]. We can see this in our testing, where every major browser engine is vulnerable in some form.





**Figure 3.3:** The fingerprint using the ‘Favicons’ exploit, explained in 3.1.1, retains its value in Chrome. The same fingerprint is generated inside the incognito window and in a normal browser window.

### 3.1.2 Cache Control

Relating to 3.1.1, we get a decent understanding how any fingerprinting vector using regular resources works.

As said before, the Cache Control header tells browsers (and other agents, such as proxies) how to cache the given resource. As website owners often control their own servers, they can tell them what to put in these headers.

We abuse the cache principle by putting resources in cache, or to tell the browsers explicitly not to cache them. To avoid cache collisions, we instruct the header that proxies (and anything other than our user agent) that the resource should always be fetched from the origin.

To prepare, we create several resources. These could be either CSS, JS, HTML or images. They might even be the ones already deployed to the website. We then assign one resource to be our *new visitor* resource. This resource will always be cached by the browser. The other resources will be our *identifier resources*

When the server receives a request for our website, we return the HTML as we normally would. We instruct it to not be cached, as it ensures that the webserver knows when a user is visiting. We then wait which other resources are requested.

We don’t reply to any identifier resources that are requested before we either receive a request for the new visitor resource, or a timeout occurs and we’re relatively sure that the user has the resource cached.

In case the user requested the new visitor resource, we know that they didn’t visit our website before. The reply to it with the resource and the instruction to cache this value. We then generate a binary identifier and assign the other resources either to cache or not to cache the resource, according to whether the identifier bit is 1 or 0.

Corresponding to the favicon fingerprinting vector, when the main page was requested, but the new visitor resource wasn’t, we can safely assume that this isn’t the first visit. We then listen which requests to our fingerprinting resources arrive. Those will be the 0-bits in our identifier. By filling in the non-requested fingerprinting resources with 1-bits, we can retrieve our identifier.

The big advantage of this approach is that it wouldn’t require any redirection, as was the case with our favicons. Also, it uses just core HTML functionality (i.e. linking),

which has a more essential functionality than the favicons (favicons could be just ignored to fix the previous attack).

It is unfortunate that such an attack would be easier to mitigate, as these resources would be flushed as soon as the browser is instructed to do so. But, by making a slight adjustment we could even detect when the browser had its cache cleared: by making the favicon our new visitor resource, making use of the favicon vector.

### 3.1.3 Alt-Svc

While the previously discussed methods work as expected, other vulnerabilities exist. One of these is the HTTP Alternative Services header. This header indicates to browsers that the provided website is available by other means [49].

The header plays an important role in the roll-out of newer protocols, for example the new HTTP/3. By using this header, servers can provide (possible experimental) HTTP/3 implementations on different port numbers than other implementations [76].

The fingerprinting potential is twofold. On one side, the header provides a timestamp for how long the alternative source should be available [76]. This can be used to put clients in individual buckets and thus assign them a fingerprint. As browser will subsequently connect to the alternative service (if available).

On the other hand, this header could be used to perform a port-scan and detect which firewall rules might be applied on a host device [76]. This could be done by having a subdomain for every port number that we want to scan. While it is impractical to scan all  $2^{16}$  available ports, ports can be selected to focus on common application ports and several *arbitrary* ports to check whether the firewall were to block such outgoing connections.

### 3.1.4 Mitigation

A first and best mitigation would be to eliminate the cache completely. If websites cannot deduce anything from the cache, its fingerprinting potential vanishes.

The Tor browser doesn't keep any cache, for the same reason no history is stored. As this cache could be used to reverse-engineer which websites were visited on a given installation. Therefore, vulnerabilities concerning the cache don't seem to affect this browser.

But, this would drastically increase bandwidth use and, following that, page load times. Websites rely heavily on the browser cache to optimise the experience for users. Also, bandwidth towards data centers worldwide would increase if every request would require every resource to be reloaded.

A balance between performance and privacy could be to keep a cache per domain. While this wouldn't eliminate the possibility of fingerprinting, it would limit the fingerprinting to the website itself. One disadvantage is that a lot of websites use JavaScript and CSS libraries, which are hosted on a CDN. If these had to be downloaded by every website a user visits, it would still provide a big overhead. Also, storing these separate on disk would yield more disk space. However, that disk space could be reduced by sharing the files that multiple websites have cached (and

storing only one copy). This approach is what Firefox recently shipped in their 85th version [24].

Another compromise would be to often clear the caches and delete all their contents. This would give the advantages of a cache in the short-term, and eliminate the long-term identification. However, it would still allow for some cross-domain fingerprinting to be conducted. Therefore, in our opinion, the previous approach would yield better results.

One thing that isn't considered, is the use of metered connections. As bandwidth is very expensive in some developing countries, the cost of this extra bandwidth should be considered as a disadvantage to this cache protection.

### 3.1.5 Conclusion

The cache is a big browser fingerprinting vector. By its very function, it keeps device-specific data.

Mitigation seems rather difficult, without either impacting bandwidth, disk space or loading times. Otherwise, programmatic access to elements from the cache should be disallowed, which has historically proven difficult.

## 3.2 HTTP stack

Many versions of the HyperText Transfer Protocol exist. While the original 1.0 is still supported, many clients use version 1.1 as their default.

While HTTP can be considered a network element, we put it in the browser-based fingerprinting vectors. The reasoning behind this is that the HTTP stack is implemented in the browser, as oppose to e.g. the TCP/IP stack, which is often implemented in the operating system.

In this section, we explore the different versions of HTTP and the fingerprinting that the particular versions of the protocol enable.

### 3.2.1 HTTP versions

The first and most obvious fingerprinting vector would be availability of different versions in browsers. All major browsers support HTTP/1.1 and HTTP/2. While HTTP/3 is still in active development, an increasing number is supporting HTTP/3 as well. About 19% of websites support HTTP/3 in some form [79].

While we could not find any statistics about how many connections are served over each HTTP version, we assume that the data is closely related to other data in the browser fingerprint, such as the browser and its version number. Therefore, it could be not that interesting to fingerprint.

HTTP/1.1 was the first version of HTTP to be standardized in a RFC. As most web-servers and webbrowsers support it, initial requests to websites are often made using this version of the protocol.

### 3.2.2 HTTP Headers

The main differences between user agents in their approach to HTTP, is the use of different headers.

The most obvious header is the User-Agent header, which gives an indication for the browser and version to a server. It was introduced to allow website administrators to have better versions of their website for different browsers.

User-Agent isn't the only header used for fingerprinting purposes. Another one is the Accept-Language, which indicates to a website which languages the user of the browser can understand. This header helps in cases where a website supports a variety of languages. Without this header, some sort of language selection page could be shown.

Ironically, the 'Do Not Track'-header is used to fingerprint as well. As mostly privacy-concerned people enable the header (all major browser disable the option by default), having the header present can be a fingerprinting disadvantage.

Cover Your Tracks, the successor to Panopticlick, generates statistics about how seldom certain values are seen. The DNT header appears in one out of every 2.06 browsers [2]. However, this number shouldn't be considered representative, as the visitors of the website are mostly aware of fingerprinting practices and are thus more likely to have protection against web tracking installed.

Numbers from Gizmodo Media Group indicate that somewhere between 6.7% and 10.9% of their visitors had DNT enabled in 2018 [31]. While still being a niche audience with relative older numbers, it gives a better indication as to which degree DNT can help to fingerprint more clearly. We weren't able to find numbers for websites with a higher and more heterogeneous audience.

### 3.2.3 HTTP/2

Being the first major revision of the HTTP protocol, HTTP/2 gave major improvements to efficiency and speed. Headers are compressed, multiple streams can exist at the same time and without using multiple sockets. To promote the use of encryption, major browsers only support it on HTTPS traffic (except for loopback connections).

Several elements of the HTTP/2 protocol are fingerprintable. Besides the headers from HTTP/1.1, HTTP/2 introduced streams. These streams can have different window sizes. As the values differ per browser, these values are fingerprintable [65].

### 3.2.4 HTTP/3

At the time of writing, HTTP/3 is still in active development. Many browsers and websites have implemented experimental implementations to be able to provide better insights to the continuing development of the standard. Chrome and Firefox both have implementations, but only chrome enables the functionality by default.

The HTTP/3 development began after Google submitted their in-house developed competing standard 'QUIC' for standardization to the IETF.

gQUIC<sup>1</sup>, QUIC and HTTP/3 are terms that are often used interchangeably, but are

---

<sup>1</sup>Google QUIC

not exactly the same. QUIC is the name for the underlying transport protocol used by HTTP/3. It was chosen to develop QUIC on top of UDP rather than develop a new protocol, as routers and firewalls historically only had TCP and UDP to be available and might block a new protocol [57]. Nevertheless, QUIC implements almost all of the TCP features itself. gQUIC is the name of Google's implementation of the QUIC protocol and the HTTP that runs on top of it. While becoming mostly compliant to standard, some minor differences exist to this day. The biggest difference between QUIC and gQUIC is in its cryptography support. Google deployed their own encryption, whereas the IETF-approved QUIC uses TLS 1.3 [57].

### Transport-layer fingerprinting

One problem the QUIC protocol was designed to solve, was so-called Head-of-line blocking. This problem manifests when we miss one TCP fragment of a stream. While we might be able to get Recent research claims to be able to be able to conduct website fingerprinting. By applying machine learning models to guess which traffic a website is coming from, it should be possible to deduct the exact website a user tried to visit [91].

While [91] doesn't directly address browser fingerprinting (the authors focus on implications for censorship), when combining which websites are being visited, we might be able to get which user we have. Although, such fingerprinting would require a massive amount of websites to have been trained in the model. Also, since lots of people browse to the same subset of Alexa 100 websites, it might not be feasible to use this vector to fingerprint individual clients.

### 0-RTT

A major improvement of QUIC (as oppose to HTTP/3) is the ability to serve existing clients without handshakes. Here, encryption parameters are negotiated on the first visit. Subsequently, the browser remembers these parameters for use in future session. This concept is called *0-RTT*, or zero roundtrip time. It alludes to the lack of additional roundtrips that would be necessary in other HTTP versions.

While 0-RTT reduces the time to load a (well-optimized) page significantly, it also provides a new fingerprinting vector. As security requires each client's key to be unique, it can basically be used as a unique identifier [73].

The additional problem this imposes is that the parameters are server-dependant, meaning that the vector could be used in third-party cross-domain tracking [73].

As configuration can decide how long 0-RTT tokens can be reused, the usability of this fingerprinting vector becomes implementation-specific [73]. This is advantageous in the protection provided by clients, as these can keep the values low to protect them from long-time fingerprinting.

As the parameters for 0-RTT fingerprinting are stored inside the user agents, mitigation's for the 0-RTT fingerprinting vector are similar to those of caching, discussed in 3.1.4. Mainly, the issue is the fact that the browser stores unique identifiers that are used later on. As with the caches, the potential for third party fingerprinting can be resolved by storing these values per website.

### **Congestion control**

As HTTP/3 is based on the UDP protocol, which only adds port numbers and a checksum to the network layer's Internet Protocol, elements such as congestion control, handshakes and acknowledgements had to be re-introduced.

One of these things, congestion control, has had several big developments in the past years. Several TCP versions gave new congestion control algorithms, as can be seen in 5.5.

The reason we discuss QUIC congestion control here and not in chapter 5, is that QUIC is implemented in user space (i.e. the browser), whereas TCP is implemented on OS-level (e.g. the Linux kernel). Changing the congestion algorithm requires kernel parameters to be changed, whereas every browser using HTTP/3 can decide how their implementation handles it. Several developers have stated that this was a deliberate action to be able to keep pushing newer revisions, without being blocked by an older OS-level implementation. This would allow QUIC to respond to newer and better congestion control algorithms in a quicker way.

### **Available drafts**

As HTTP/3 is still in development, new drafts are often released. These new drafts introduce changes to the protocol and are incompatible. Both client and server have to understand the same draft version to ensure compatibility.

To increase the likelihood of a compatible draft version with different clients, web servers understand several drafts and can thus pick their HTTP/3 implementation to match one that the client supports.

This concept might seem similar to the HTTP upgrade header, which allows connections of HTTP/1.1 to be upgraded to HTTP/2, and the similar concept in TLS, where servers can indicate which HTTP-versions are supported, even before the first HTTP request. A big difference with the HTTP/2 upgrades, is that there is a greater variety of available drafts to choose from, increasing the fingerprinting potential.

Over time, as HTTP/3 stabilizes and the drafts become a formal specification, this fingerprinting vector will become less useful (and more in line with the HTTP/2 upgrades). Browsers will drop support for drafts of the specification and only support the one published in an RFC. Drafts have expiration dates as well, meaning that they shouldn't be used after a small period of time.

#### **3.2.5 Extensions**

Most mainstream browsers allow the installation of extensions (or add-ons) to be installed by their users. These extensions make browsers behave differently, add new functionality or allow certain hardware to communicate better with web applications.

As these extensions are browser-specific, potential to fingerprint them is available.

Extensions need to be able to show users options as to how to configure them, login to their service or to add code to existing webpages. To solve these problems, extensions are able to create their own webpages, which can be shown in the browser as a normal tab or a pop-up style view. As these webpages behave as normal webpages,

there is a need for a URI to access these resources. Accessing these resources is often done using a special extension-protocol (e.g. `moz-extension://`). If a website were to make a request to a url and recieved a 2xx-code, it could be sure that the extension is installed.

Websites can no longer see which extensions are enabled with absolute certainty (at least using the previously discussed method). However, this doesn't mean that no fingerprinting is possible.

The most popular category of extensions consist of content blocking [3]. As this category contains so-called adblockers, its main goal is to block content that users don't want to load, mainly advertisements. While applications exist to avoid advertisements on network layer (in the form of a DNS server), having the option in the browser allows for better results. As websites can include ad-related code on the same domain as the main content of the website (e.g. YouTube), meaning that a DNS-level blockage would not work. As these extensions are able to check and block each individual request, the amount of ads that can be blocked is significantly higher.

Content blockers have other uses as well. Privacy Badger, for example, tries to block third parties that seem to track users across the World Wide Web. Not relying on a hardcoded list (as is the case with many adblockers), it uses heuristics in terms of when it sees a given domain. This makes it more difficult from a fingerprinting perspective to find out whether the plugin is installed. But, on the other side, the fact that not all domains are treated equal on each client could be used as a fingerprinting vector itself. But, these values should not be considered stable, as it could be blocked if the extension decides that too much external websites are calling it. Also, when it isn't sure yet, it can only block cookies. Therefore, we do not see Privacy Badger fingerprinting as stable.

Schemes exist to detect whether content has been blocked. A website can e.g. detect whether a scripts was loaded and then show a popup.

Other forms of extensions exist. According to Mozilla Firefox Add-ons<sup>2</sup>, the most popular ones include password managers, additional functionality for specific websites and download managers [3]. The tendency towards privacy-enhancing extensions might not be entirely representative for all browsing users, as Firefox promotes itself heavily as a privacy-enhancing browser.

Fingerprinting of these other categories of extensions is fingerprintable by different means. JavaScript code can e.g. check which DOM nodes were added (e.g. password managers showing a button to autofill credentials) or even user-agent spoofers (by detecting inconsistencies in reported values) [10].

### 3.2.6 Plugins

Another element that works similar to browser extensions, is browser plugins. The difference is that these plugins are able to provide completely new functionality, as oppose to altering existing browser components and technologies.

The difference between plugins and extensions is in their implementation. Extensions are mainly programmed using traditional web standards (HTML, CSS, JS). Plugins are written in natively compiled languages, such as C or C++.

---

<sup>2</sup>Google Chrome Webstore doesn't have a list of most popular extensions



Historically, NPAPI plugins were dominant on the web. Recently, however, the main focus for these plugins went to DRM<sup>3</sup>-solutions, such as Google's Widevine. To detect the availability of DRM solutions or other plugins, we can try to initiate their functionality. In the case of DRM, we can try to play a media file using the DRM. If this were successful, we know that a working setup exists with the DRM solution enabled [47].

### 3.3 NPAPI

In the past, lots of interactive applications relied on the use of NPAPI plugins. The most popular plugins were Adobe Flash, Java and Silverlight. These plugins allowed for more native applications to run in a browser, providing functionality previously not available to websites. Because these APIs gave more native access, they could open up access to native components, such as the GPU, the file system and webcam.

The two main vectors for Flash-based fingerprinting were font probing (similar to 3.4) and system information querying (which was built-in into a Flash API call) [36].

While HTML5 provides similar functionality to Flash, the latter gave slightly different results when used in a fingerprinting context. One such example is the Platform variable on Linux. Where a browser would usually expose just "Linux\_x86\_64", Flash would give websites the exact kernel version [78].

Most browsers at the time of writing have dropped support for NPAPI, making it impossible to run them at all. The main reason for the deprecation is the amount of high-risk vulnerabilities that have come with the API. As the API was introduced in the 1990s, with fewer security concerns compared to today. While outdated versions of these browsers (or versions with long-term support) will remain functional for now, continuing to use these will become more and more discouraged.

While NPAPIs provide a variety of possibilities from a fingerprinting perspective, their deprecation makes it infeasible to ever use them again on large scale fingerprinting in the future. The amount of browsers still offering support for it will only decrease, and those versions that do support it, will lose market share. However, it will likely remain a distinguishing factor between newer browsers and older ones (or browsers that won't drop support for the API).

While NPAPI is getting deprecated, the fingerprint potential remains. Where Flash and Java could probe to fonts and webcams using their native access, the availability of webfonts and HTML5 Webcam access respectively. The only aspect that will remain as a fingerprinting vector, is whether NPAPI applications can be run in a given setup.

### 3.4 Font probing

Different systems can have a different set of fonts installed on them. Certain applications, such as Microsoft Office, install extra fonts when they're installed. Some business require special fonts for their branding. Therefore, a variety of fonts can be installed on a single system.

---

<sup>3</sup>Digital Rights Management. Technology to prohibit unauthorized use of digital media.



One element that can be different between systems, is the list of installed fonts. While we get this information immediately using an API, we can get a list of fonts of which we are certain that they are installed [74].

To check whether font  $x$  is installed on our system, we include the following rule in CSS:

```
font-family:  $x$ ,  $y$ ;
```

We then provide font  $y$  as webfont. If we see the browser make a request for font  $y$ , we can safely assume that font  $x$  wasn't available on the system.

A mitigation for this attack vector would be for browser to not use system installed fonts, or to only use a fixed subset of them, which would then be a universal set of available fonts on the web.

As mobile devices are having the biggest market share, it should be noted that the effectiveness of this vector is diminishing. Android smartphones in our testing often only have the Roboto font at their disposal.

### 3.5 Conclusion

In this chapter, we saw that a lot of fingerprinting is possible using just the browser and the elements that it exposes. Mitigation for these attack vectors can be difficult, or have unwanted effects. A summary of the techniques seen in this chapter can be found in Table 3.1. Here, we also examine the values for our criteria.

Metric / Vector	Browser Cache	Favicons	User-Agent	DNT
Degree of entropy	N/A	N/A	Medium	Low
Amount of extracted data	N/A	N/A	8 bits <sup>a</sup>	2 <sup>b</sup>
Stability of the result	In-Browser	In-Browser	In-Browser	In-Browser
Impact of complete Mitigation	Functionality	None	None	Functionality
Speed of execution	Instant	Slow	Instant	Instant
Impact of execution	None	Redirections	None	None
Percentage of users vulnerable	>99%	Min. 84.5% <sup>c</sup>	>99%	>99%

Metric / Vector	Accept-Lang	Alt-Svc	HTTP2 stream defaults	QUIC 0-RTT
Degree of entropy	Medium	N/A	Low	N/A
Amount of extracted data	N/A	2 <sup>16</sup> · x <sup>d</sup>	N/A	N/A
Stability of the result	OS <sup>e</sup>	In-Browser	(In-)Browser	Browser
Impact of complete Mitigation	Functionality	Functionality	Performance	Performance
Speed of execution	Instant	Fast <sup>f</sup>	Slow	Instant
Impact of execution	None	None	None	None
Percentage of users vulnerable	49.6% <sup>g</sup>	>48.9%	45.5% <sup>h</sup>	Max. 80.3% <sup>i</sup>

Table 3.1: Comparison of different browser-based fingerprinting vectors

<sup>a</sup>estimation: browser (4 bits) + version of browser (4 bits)  
<sup>b</sup>Can be given or not, can be enabled or not  
<sup>c</sup>No exact number could be found of browsers that support all favicons. This number is only the amount of clients that support PNG favicons.  
<sup>d</sup>Each bit can be a port that can be communicated with. x is the amount of available IP addresses  
<sup>e</sup>Language should be changed  
<sup>f</sup>Depending on behavior of browsers when a port is blocked  
<sup>g</sup>Numbers from caniuse.com  
<sup>h</sup>All major browser clients support HTTP/2. Only 45,5% of websites support it [79].  
<sup>i</sup>Amount of clients with HTTP3 support.



## Chapter 4

# Exploits using Web APIs

### 4.1 Introduction

In the previous chapter we saw how we could exploit vulnerabilities that are linked to the browser. In this chapter, we explore the different Web APIs that are available.

We explore some APIs that seem relevant for our practical application later on, where we try to use fingerprinting in a game environment.

#### 4.1.1 W3C Web Standards

The W3C consortium, which is the body responsible for the development of these standards, has decided to add a Self-Review Questionnaire for developers of standards to fill in and attach as integral part of the standard. This questionnaire lets the developers themselves think about the possible fingerprinting vectors of their new standards, it gives a more uniform way to identify issues. Also, by letting the authors of standard think about fingerprinting, awareness of the topic is raised, allowing authors to think about the subject when continuing to write standards. It is recommended to fill the questionnaire early on in the development process, as some problems might be found after the standard has been shipped in browsers. When catching potential problems early on, it is easier to change the design of a particular API [7].

Off course, filling in the questionnaire doesn't mean that problems surrounding fingerprintability are solved. The questionnaire only lists the findings of the authors and they are not required to mitigate the found vectors. As we will see shortly, there are many cases where reducing fingerprintability would do serious harm to the purpose of the API. In many cases, developers just acknowledge that on privacy-critical applications, such as in e.g. the Tor Browser, access to certain parts (or the entire) API should be either omitted or (in cases where hardware access occurs) be software emulated [86] [8].

#### 4.1.2 Environment

In the previous chapter, we saw different ways to identify the browser and its configuration. However, one easy way to completely bypass these fingerprinting mechanisms would be to download a different browser. Besides that, opening incognito would be an easy way to change some browser behavior, as browsers tend to disable some extensions and APIs in incognito mode [70].

But, even that could not be enough, as browsers which previously had their own engines have switched to using the Blink engine, made by the Chromium project. These browsers include Edge and Opera. In a lot of scenarios, they actually get a similar or identical fingerprint, as can be seen in 4.2 and 4.3.

When looking further than the realm of a browser, lots of new APIs are available that offer more direct access to the underlying hardware components of a system. Can these be exploited to fingerprint certain elements of the operating system, or even underlying hardware?

In this chapter, we explore the different ways in which we can exploit the available APIs to fingerprint other components of the user's system. Because *system* could be a vague notion when talking about these components, we define the term *environment* as the combination of the available hardware components in a system, as well as the accompanying software. Under this software, we consider the operating system and its closely related components (such as drivers to use the hardware) and software that have a mere utilitarian role (such as fonts).

Please note that the environment does not include anything related to the network connection of the device (except for the NIC). These fingerprinting vectors will be discussed in 5.

Fingerprint vectors that are based on elements from the environment could be considered more effective, as these vectors require more actions from the user to protect themselves. Where users could get past browser-based fingerprinting by installing a new browser, some environment vectors might require an actual hardware swap, which makes it more difficult.

### 4.1.3 GPU

A big part of this chapter will be dedicated to fingerprint attack vectors centered around the GPU. Traditionally, direct access to the GPU was reserved for native applications, such as media players<sup>1</sup> and videogames. However, within the last 10 years, several new APIs have risen that give web developers more direct access to the hardware, laying the path for the web as a feature-rich platform. These APIs include the Gamepad API [28], WebXR [88], Websockets [87] and Vibration API[82]. For GPU use, the most prominent one at the time of writing is WebGL, which brings an OpenGL-like API to websites.

Nevertheless, a new API (called WebGPU) is in development. Whilst WebGL was a standard by the Khronos Group, WebGPU is a web standard by the W3C. It is set out to be backend-agnostic, with support for i.e. WebGL, OpenGL ES, Vulkan, Metal and Direct3D [85]. As this API is still in active development, not a lot of research is done around this.

## 4.2 HTML5 Canvas 2D font rendering

The HTML5 specification introduced the canvas element. According to its specification, it provides a general purpose graphics element. When using the element, developers can opt for several drawing contexts, which correspond with different implementations in browsers to use the <canvas>.

---

<sup>1</sup>A lot of GPUs also offer video hardware acceleration.

In 2012, researchers found that the HTML5 canvas could be used as fingerprinting vector [44]. The experiment in [44] focused on font rendering on the canvas. The reasons that this gets some entropy are twofold. One aspect is that aliasing is set differently in different browsers, which introduces some variants among browsers. Another aspect is that fonts could be available or not on a given system, meaning that we could get a fallback font if the desired font is unavailable (increasing entropy).

#### 4.2.1 Experiment

When we tried to replicate this work, we were able to conclude that this fingerprinting technique is still able to differentiate clients. By simply drawing several fonts in different colors below each other, we could get a fingerprint.

However, we noticed that we got the same fingerprint on two Linux-based devices running the same version of Firefox. On Chromium, we got unique fingerprints for both devices. This could be due to the GPU compositioning that Chrome uses; Firefox uses a software-based alternative [30].

We enhanced our experiment with an extra call to a default font (sans). We do this because systems could have a variety of default fonts, especially on Linux.

On Windows and Linux, we got better results by not following the paper and having some fonts not available for drawing. This increased the amount of potential unique outcomes, as users can have different default font entries.

One problem we detected was that running the experiment on two identical Android smartphones (same model and same browser), we got the exact fingerprint. The reason for this seems to be that both phones have the exact same GPU and font (only Roboto<sup>2</sup> is available), as installing fonts on a smartphone is more cumbersome than installing fonts on a desktop operating system.

So, while the experiment still holds up, with the rise in popularity of smartphones, with their limited amount of available fonts, using the rendering as a fingerprinting vector, does not give a unique fingerprint.

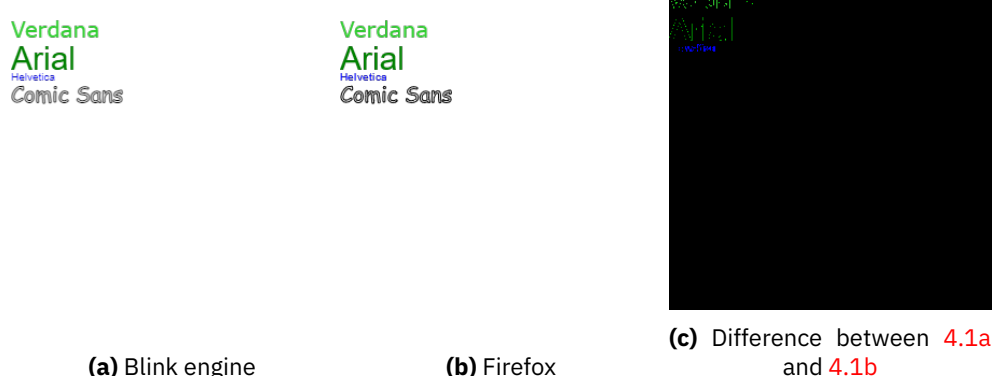
One way to address these concerns would be to use webfonts to fill the missing fonts. But, as the smartphones already give unique results, and input to both devices would remain equal, we do not expect this to make a difference. Besides, the webfonts would take away entropy caused by the different availability of fonts on the system.

### 4.3 Canvas 2D drawing without fonts

The authors of [44] could not identify an approach to fingerprint using a technique other than fonts. We do a small attempt at it. By drawing a shape with a gradient as background and another gradient as its contour, we were able to extract some entropy out of the canvas 2d, but as with the fonts, we were not able to distinguish between two Firefox instances on Linux. So, we do not see this as a viable option.

---

<sup>2</sup>Font developed by Google mainly for its use on Android.



**Figure 4.1:** Our experiment as rendered on Windows.

## 4.4 WebGL

While the canvas fingerprinting vectors of 4.2 can distinguish users, we can get more variation between users by gaining closer access to the actual GPU hardware.

WebGL, as mentioned in the introduction, has several ways to extract entropy out of it. In this section, we follow the use of the vendor information, found in the WebGL Debugging extension. This data contains both the model number of the GPU that is used for rendering and the manufacturer of this GPU.

### 4.4.1 Experiment

We created an experiment that tried to extract the name of our GPU and its vendor. We were successfully able to identify the dedicated NVIDIA GPU in a desktop computer, as well as the integrated GPUs in laptops and the mobile GPUs in smartphones.

On Windows, we noticed both Firefox and Google Chrome reported the ANGLE vendor. This appears to be due to the usage of Direct3D to draw the graphics. Both browsers use ANGLE, a backend by Google that translates the WebGL calls into Direct3D calls. In the renderer field, we could still see the model of our GPU. The two browsers use ANGLE by default on Windows [13].

On Linux, we see that Firefox used OpenGL calls directly to the drivers, whereas Blink-based browsers (Google Chrome, Chromium and Microsoft Edge) used the ANGLE backend on this platform as well. ANGLE can use different backends, making it the perfect cross-platform cross-API layer for WebGL. This does mean that we have a fingerprint collision on these browsers [13].

### 4.4.2 Mitigation

Several mitigations can be found [44].

One could be to have a large number of clients return the same value (masking real values). While this could make the fingerprints less unique, it could make differences between browsers more noticeable.

But, when doing this, we might as well just omit the entire extension. As this is a WebGL extension, browsers can choose not to support it and, when applying best

practices, developers should have always assumed that these extensions might not be available. However, it would be inevitable that some applications were not built according to this principle and might break, as disabling the API would be a breaking change.

Firefox will not enable the WebGL debugging extensions when using its ‘resist fingerprinting’ option, making the vector unusable. As the Tor Browser enables this option by default, the vector does not affect these clients. However, as with many vectors, the absence of data could be used to fingerprint.

#### 4.4.3 Conclusion

Extracting vendor information is easy to implement, making it useful as a *quick and dirty* way to extract some entropy out of a WebGL client. Furthermore, as the vendor information is probably closely related to other extracted data from WebGL, it might not be a very interesting to use. It could still be interesting to find out whether vendor information can be linked to render artefacts, which could allow the vendor to be predicted from the canvas. Especially with the rise in machine learning capabilities and research, this could mean that hiding the debugging headers could not serve as enough protection.

The vector has nearly no impact on system performance. It does require the availability of a canvas element, but could reuse anyone that was on the page before. Also, it can be integrated in e.g. the WebGL Pixel Buffers (see 4.5) vector. Or, if the page doesn’t have a canvas element, the element doesn’t even have to be added to the DOM.

The available range of values is rather low. As there are only a relative small amount of available vendors and models of GPUs. Moreover, when used to e.g. distinguish devices that are e.g. behind a business’ NAT, it might not be feasible to use just the vendor information, as these devices could be bought in bulk and have very similar configurations.

Using the vendor information could be considered stable, as changes to the configuration would have to happen before a change in fingerprint can be detected. This configuration could be either a making modifications to the driver, booting into another operating system or changing out hardware.

But, mostly due to the lack of entropy, we don’t see the vendor information field on its own as a big fingerprinting vector.

#### 4.4.4 WebGL Debug Shaders

Another small fingerprint vector in the WebGL Debugging extensions could be the debug shaders. This will return the source for the shaders as compiled by the graphics driver [84]. This could be a viable option, as we just saw discrepancies in backends used by the two main browsers.

Because several browsers use different WebGL backends (as discussed before), their compiled shaders differ slightly. We could find that differences could be seen between Firefox and Chrome on the same machine. Maybe this principle can be extended to the other part of the WebGL debug extension?



#### 4.4.5 Experiment

According to MDN, Firefox should require an `about:config`-setting change in order for this function to work [84]. However, we were able to exploit this without any changes to the configuration. We even tried this on a clean Firefox installation and were still able to extract a value.

There's no support in IE. All other major browsers do not require any configuration for this feature to work, meaning about 90% of all browser support this feature without further user interaction.

#### 4.4.6 Mitigation

Whereas mitigations for the other values in the debugging extensions could be mitigated by giving out uniform values, this principle cannot be used for the debug shaders. These values depend a lot on the particular shaders that are given as inputs.

The most convenient ways to mitigate these fingerprint vectors, would be to simply disable the debugging extensions by default and require changes to the configuration of the browser in order to use them.

Another way to mitigate the vector would be to

#### 4.4.7 Conclusion

Our experiments show that the vector can be exploited without user permission in any browser that supports the WebGL Debugging Extensions.

It was possible to extract entropy from the debugging shaders. However, we assume that the gathered entropy will be rather limited and closely related to other WebGL techniques. These other techniques, such as 4.5, will be more effective.

Also, the vector is not practical as additional vector in an existing and 'legitimate' WebGL project, as it depends on the shaders not being changed, which puts limits on the abilities of projects.

We're unsure about the stability of the vector, but assume that these can change in between browser versions, as the ANGLE-layer will most likely receive an update as well.

### 4.5 WebGL Pixel buffers

The most common way to exploit WebGL as a fingerprinting vector, analogous to the 2d context, is the use of data in the pixel buffer. While the workflow is similar to the 2d context, it gives more entropy, as we're working closer to GPU hardware, which involves more floating point operations.

In 2012, it was found to be possible to fingerprint using WebGL [44] Where the previous canvas methods mostly relied on font rendering, WebGL fingerprinting mostly relies on the aliasing of rendering textures or noisy shapes on top of each other [11].



**Figure 4.2:** The texture used to create some entropy.



**Figure 4.3:** The texture of 4.2 projected onto ‘Suzanne’, the monkey shape in Blender [15]

### 4.5.1 Experiment

We wrote our own experiment, based on [44] and [11]. Where [44] uses a lens testing image, we thought it might give better results when using a completely random image as a texture, which we subsequently created. This texture can be seen in Figure 4.2. We projected it on the monkey shape baked in into Blender, the result of which can be seen in Figure 4.3.

### 4.5.2 Mitigation

Several resolutions to this fingerprinting vector exist. One would be to add noise to the output of `ReadPixelBuf`. While this could help, fingerprinters could run the test multiple times and try to find the normal distribution of the noise, which would require browsers to add more noise, which in turn could start to hurt performance and even usability of WebGL applications.

Another mitigation consists of completely disable any way to read the pixel buffer, a technique which is used by Firefox when enabling `resist_fingerprinting`. This will ask users whether a script is allowed to read the buffers, indicating the potential dangers. The question remains whether people will be tempted to press ‘Allow’ when they see that the animation, game or other potential WebGL component has suddenly stopped working.

Third mitigation would be to simply disable WebGL or render everything on a software level. This is the current approach of the Tor Browser. It doesn’t allow calls to the two methods that are able to extract pixel data.

A fourth way to mitigate the problem would be to find the root of discrepancies between renderers. Researchers have been able to do this successfully. It consist of moving all floating point operations to the JavaScript level instead of in a shader. It does come at a price in terms of CPU usage, but it actually reduces load on the

GPU, as the shaders are more straightforward [90]. However, as this solution allows arbitrary shaders to be used, it is possible for shaders to exploit other potential weaknesses in the WebGL API.

Finally, a software implementation could be used to calculate the pixels only when a call to `ReadPixelBuf` or `toDataURL` was done. While this approach would indicate a huge performance loss whenever one of these functions are called, it would only have an impact when these are called [44].

## 4.6 WebGPU

As mentioned before, WebGPU should be seen as an abstraction on top of APIs like WebGL. It uses multiple backend APIs in order to be a more universal API.

Because it uses several backend APIs, calls to a specific API (such as the OpenGL backend) will not be guaranteed to be working. Therefore, existing fingerprinting vectors such as those found in WebGL, might not be applicable.

However, we can still try to fingerprint using a pixel buffer. But, what could be the influence of the multiple backend APIs? After all, a driver could offer multiple APIs, e.g. both OpenGL and Vulkan on a Linux system. Which one the browser chooses, might depend on multiple factors and could not be predictable.

The authors of the standard did consider some fingerprinting vectors while making the standard [86].

**Machine-specific limits** The current draft of the standard has several limits as to what usage of the GPU is allowed. Not only does this withhold exhaustive use of resources (as to e.g. detect the amount of VRAM), it also sets limits as to what performance can be expected from a WebGPU implementation, as all implementors should support them. These limits include things such as the maximum amount of vertex buffers, the maximal size of textures, etc [86].

However, powerful systems could choose to put these limits higher, as the limits are just recommendations [86]. While developers should assume the worst case settings (being the limits set in the standard), they could e.g. try to load bigger textures to see whether a crash occurs.

As the standard is yet in development and

**Machine-specific artifacts** This attack vector is similar to the WebGL pixel buffer attack vector, as discussed in 4.5. While they do acknowledge it, their response is:

Privacy-critical applications and user agents should utilize software implementations to eliminate such artifacts [86].

This is roughly the equivalent of the third discussed mitigation in 4.5.2.

**Machine-specific performance** This is benchmarking. The development team indicates that it would be possible to measure performance based on the speed at which workloads are executed, even if the values could be seen as low-precision.

The gpuweb workgroup, which makes the standard, proposes that browsers should eliminate the link between GPU calls and actual compute units, making the speeds impossible to determine. While this approach seems feasible, there could potentially be more ways to find out what performance was received.

## 4.7 Web Audio API

The Web Audio API is developed with low-level audio creation in mind. While this functionality could've been implemented in JavaScript, there are some advantages to having the functionality available in a web standard. The main advantage is that latency can be much lower, as the calls to functions could be translated into calls to an API written in a faster language, such as C. This would create less latency between the sound being requested and the sound being played back to the user. In time-critical applications, this could yield a significantly better user experience.

The authors of the standard itself state several attack vectors.

The first attack vector is the processing latency. As the API is meant to be used in latency-critical applications, API calls exist to retrieve the delay between the function call and the sound playing.

Another attack vector that the authors recognise is the fact that hearing tests could be deducted from this API. While this could be a reliable way to identify a person, the available bits of entropy (being frequencies that the user is able to hear) is rather low. Also, this vector would require active user participation, which would make it impractical in a deployment where users don't have time. But, it might be of interest in our game context, where we would be able to add a gaming element when a sound is heard.

The third and most audiophile attack vector comes from the actual sound software (and even hardware). Also, their sample rate can be fingerprinted.

But, the software component isn't immune to fingerprinting either. These (mostly) software aspects can give small discrepancies in sound, due to the use of different compilers with various compilation optimization flags.

### Mitigation

The standard discusses different means to reduce the fingerprinting surface.

Similarly to the WebGPU standard, attempts are made to reduce the fingerprinting surface by making not all native functionality available, but only the most common use cases. One such example is the proposed limitation to stereo sound. Another such example is the reduction in available frequencies for browsers to allow. Where sound devices can work on a big amount of sample rate frequencies, the Web Audio API advises to only use two main ones. While this is a valid approach for a lot of use cases, it could introduce aliasing artifacts. For these approaches, the standard recommends to have a 'setting' in browsers, where users who need the setting can change it. However, if a user enables it, it would be a fingerprinting element once again.

Finally, browsers are encouraged to find means to protect users against fingerprinting.

## 4.8 Performance API

The Performance API allows websites to gain information about the strength of a computer in terms of computing power [54]. This API allows them to perform benchmarks or get an indication as to the user experience on a given website.

As the API is meant for benchmarking purposes, it can be used to get an idea of the clock skew [61]. Using other APIs related to cryptography, native code can be run and its execution time can be measured. By comparing the execution time and delay of these functions, a fingerprint value can be derived.

## 4.9 Battery

Batteries are most commonly found in laptops and other mobile devices, such as smartphones. While their use is only to provide power to the devices they're built in, some data is exchanged with the computer (such as its percentage, lifespan and health).

W3C introduced the Battery Status API to be able to read the current status of the battery. This can be very useful in situations where a better UX can be had when a developer knows they can use more compute power. E.g. an image editing application might want to show more or better previews on actions when the user is connected to a power source or has a battery that is adequately charged.

The Battery Status API can be used for fingerprinting, as shown in [52]. Firefox had an issue comparable to the Performance API issue (see 4.8), where they showed results with a high accuracy on GNU/Linux. Chrome uses another API, making the results non-fingerprintable [52].

Chrome did not have the same issue as Firefox, because of a different implementation [51]. They did however ensure that the result would always be rounded before giving it to the client JS code.

There is another known attack vector of the Battery API. When batteries are near the end of their lifecycle, they tend to be unable to report adequate values. Therefore, earlier implementations of the API gave an error when the battery was at this stage and thus opened a fingerprint attack vector. The current editor's draft explicitly states that the API should emulate a fully charged and plugged in battery when errors occur [45].

Both of these vulnerabilities are solved in the current editor's draft of the API specification. It explicitly states that no high-precision values can be used, which addresses the Firefox vulnerability. When an error occurs, it should emulate a fully charged and connected battery [37]. Browsers generally don't allow usage of the API (or they spoof results) in incognito mode.

While this does leave room to distinguish laptops from desktop computers, the fingerprinting potential of these values have become rather low. With Firefox and subsequently Safari disabling the API<sup>3</sup>, it technically doesn't even meet requirements for W3C Recommendation, which states that two major browsers should implement it. This makes the future of the API rather uncertain.

<sup>3</sup>The API was disabled for websites. It is still used internally in Firefox.

However, studies have shown that a lot of third party websites use the API, presumably for fingerprinting purposes [51].

## 4.10 Screen

A lot of websites use so-called responsive design, which allows different users to be able to see a slightly different version of a web page. Therefore, the Screen API allows a browser to read out the screen resolution, color depth etc. The API also allows for access to the resolution of the portion of the screen available for rendering.

Of course, a lot of people use their browser maximized, which makes this approach less interesting. However, as soon as people e.g. add a bookmark bar, use a different desktop environment or do not have their browser maximized, we can extort this to detect small anomalies in browsers. Also, browsers often remember the previously set size of not-maximized windows, and people often do not resize their windows, which could make it a viable fingerprinting vector if the browser is not maximized.

This actually gives the strange behavior that it is actually better for your privacy to use your browser in fullscreen or maximized mode, because a lot of people will have the same resolution. According to the Steam Hardware Survey<sup>4</sup>, at this time, 1920x1080 is the most common resolution.

While this vector could be easy to solve (by not allowing access to this data), it would harm the UX.

The Tor browser prevents this fingerprinting by asking users not to resize browser windows.

## 4.11 VR

Over the past years, Virtual Reality (VR) and Augmented Reality (AR) have seen a surge in popularity. These applications require specialized hardware, such as headsets, motion tracking devices and controllers to be installed. And as with most new technology, developers wanted it to work in the browser.

Mozilla recently launched the WebVR API. While it was more of a proof-of-concept than a formal web standard.

The availability of WebVR and WebXR are fingerprinting vectors on their own, as only a few browsers support it. But when they support it, they will prompt the user for permission. This leads to a lot of websites asking for VR permissions, even though there are no obvious use cases on these pages. We could find an example on the Belgian website 2dehands.be, a website to buy and sell items. The website is owned by Ebay and doesn't consist of any VR element.

We did not conduct any experiments, as the WebVR standard is deprecated in favour of the WebXR standard.

---

<sup>4</sup>Steam is an online store for gaming. It might not provide a scientific measurement of screen resolutions, it gives a general idea about screen sizes.

## 4.12 WebXR

Recent years have seen a surge in availability of VR equipment and content. As its popularity rose, so did the need for a Web Standard.

An initial iteration of this web standard was the WebVR standard. This proposed standard fell short. The main shortcoming was the lack of support for AR<sup>5</sup>, as its focus had been VR in the traditional form.

The WebXR standard exposes two function calls. One is to probe whether XR in different modes would be available on this system, the second is to start a session using the method that was probed using the first function.

The obvious fingerprint here is the lack of permission requests, which could be hard to mitigate once the standard is set

While the standard acknowledges that the API could be a fingerprinting vector, they don't provide a resolution. It does however cover the

We could alleviate the attack surface of the vector by requesting permission before getting access to any methods of the XR system.

### 4.12.1 Experiment

We try to find out whether an XR device is available and which sessions would be available. Because we don't have any XR-devices available to us, we use the WebXR device emulator add-on to simulate their presence.

When emulating an attached XR device, we can see that the website is able to read out details about the connected device.

### 4.12.2 Mitigation

To mitigate, a user can try to disconnect the XR-devices when they're not in active use. This way, a browser is unable to detect them. But, this is a hardware solution to a software problem; A better solution would be to ask the user before attempting to give a list of available devices to the visited website.

### 4.12.3 Conclusion

While XR devices are not commonly available, the presence of a device can be detected. This makes users of headsets that are always connected vulnerable to fingerprinting of their device.

The authors of the API did spend a good amount of time on making sure that the API contains as little fingerprintable data as possible. However, it would've been better if the API used an explicit user permission (as given by the Permission API). As the standard allows browsers to implement consents, the full fingerprint potential would be reduced to something close to zero. However, it might be possible to measure the delay between the call to `navigator.xr` and the response (which could check whether a permission was asked or if the browser just doesn't implement the API).

---

<sup>5</sup>Augmented Reality: add VR elements to a real world (either a picture or see-through glasses).

## 4.13 Gamepad API

Traditionally, games often used some form of controller. Game consoles used their own controllers, often included with the system. These were often connected using some form of proprietary connector or wireless connector protocol.

Recently, the main console brands (XBox and PlayStation) have both switched to a USB-connector. Both have support for Bluetooth connectivity. This makes them usable on a desktop platform without the need for strange connections. Both XBox and PlayStation controllers can be connected to both Linux and Windows without issues.

To make these controllers available to the web, the Gamepad API is provided [8].

The fingerprinting potential of an API that exposes access to hardware, such as the WebXR API, is mostly in discrepancies between hardware. However, as oppose to the WebXR API, the Gamepad API only allows access to the hardware after a gamepad is touched (e.g. pushing a button or slightly moving a joystick). This feature greatly reduces the fingerprinting potential for an arbitrary site, as those sites would have to trick users into touching their gamepads.

There isn't a lot of research around fingerprinting the Gamepad API. It is recorded that the calls to `navigator.getGamepads()` are actually used to fingerprint users, but not in great detail to what extent [38]. We think this might be to detect a controller that is recently touched 'by accident', or when a small change to acceleration sensors create such a small change and thus expose the controller.

We were able to find two fingerprinting vectors. One is the mere presence of one or more controllers and their types. The API allows easy access to the type of controller.

A lot of controllers suffer from drifting issues. This means that the thumbsticks of the controllers aren't centered correctly when not being touched. The sensors get inaccurate after mild use. It could therefore be safe to assume that most controllers that are regularly used suffer from some form of controller drift. This is where our second vector originates from.

### 4.13.1 Experiment

We create two experiments. One will read out the data about the controllers that are connected to the system and used in a configuration. This way, we can gain entropy from the amount of controllers and their types. Our second experiment will involve trying to identify controllers based on their amount of thumbstick drift.

On Firefox/Linux, we observe that every controller exposes a string of both the marketing name of the product (e.g. 'Microsoft X-Box One S pad'), prepended by the unique vendor and product IDs (e.g. 045e-02ea). On Chrome/Linux, this string is formatted differently. Firefox on Windows used just "xinput" as string for the type of controller, meaning it uses a DirectX API. Chrome on Windows gave "Xbox 360 Controller (XInput STANDARD GAMEPAD)". This is remarkable, as the controller inserted was a different XBox Controller. So, both browsers use a default value for the gamepad data they received from DirectX.



We start by trying to find out whether there's a consistency in the amount drift that we can observe. We do this by performing a repetitive motion on one of the thumbsticks. Upon observing the first results, we saw that these values weren't consistent, meaning that we cannot measure its absolute drift.

To find out whether there's a normal distribution in the amount of drift we measure, we repeat the same motion several times (i.e. moving the left thumbstick to the far-left-most position and letting it go, which makes it bump back to the middle). As we have two controllers of the same type, we can see if both of them give stable results (and if those differ).

After taking 100 samples of both controllers, we can see an incredibly small difference. We do observe a difference in average values. To ensure that these values persist, we do the same test on another system.

### 4.13.2 Mitigation

As with a lot of fingerprinting methods, the best mitigation would be to completely disable the fingerprinting vector.

Drift issues might be difficult to solve, as they originate from a hardware defect. However, a lot of games have some form of drift compensation built-in, as it would otherwise register as if the user touches the controller. Browsers could filter results below a threshold and replace them by a value as if they were on the origin of the axis. So, values of 0.001 should be rounded to 0.000. Despite that, it could hurt the experience when there's a need for slow or subtle movements. A balance could be found here.

One hurdle that this mitigation introduces, is the different kind of controllers available and their different axis, as some axis might not need to be modified. One such axis is a trigger axis (for the trigger buttons). As APIs often just give the axes, it would be up to the browser to determine which values to round and which not.

### 4.13.3 Conclusion

We conclude that the Gamepad API maintains a good balance between privacy and functionality. It allows developers to read out the type of controller, which can be useful to debug or optimize an application, while not giving any data to websites that don't explicitly need it. This makes fingerprinting only accessible to websites with a valid reason for users to touch a gamepad (such as online games). And, as there's no other way to read this data without users prompting to touch a gamepad, websites cannot circumvent the restriction by asking for a permission.

The websites that do have a valid reason to prompt for a controller, do get some information. While there's not much data, the name of the controller is the most obvious data for fingerprinting. However, Windows is the most popular gaming OS, and we've just seen that all controllers on that platform return the same value for this field (in the most popular browsers). This makes this fingerprint potential rather low.

Something less intuitive is the drift issue. If we can require users to move the sticks in different directions, we can measure the differences between measurements after letting go the sticks. However, we were did not get a big difference between controllers' amount of drift.

We do take this in consideration when creating our practical fingerprint. This will require a sequence where the user is required to leave the thumbsticks as is after moving them in a particular direction.

## 4.14 Media Capabilities API

The Media Capabilities API is an experimental Web API that gives applications an indication as to which combinations of codecs, bitrates and formats would be supported on a given system [41]. It is meant for web applications that offer media streaming in different formats, and want to make sure that the user will get the optimal format for its system.

The main function call in this API is `decodingInfo`. It can be seen as a continuation and extension of the `canPlayType` function call that existed before. Whereas the latter only replied whether media was able to play, the new function call extends this information to the smoothness and power efficiency of the playback [41]. Also, the API provides calls to check the availability of HDR and related properties (such as color gamut). While no browsers currently implement these function calls, they are included in the latest version of the standard [40]. This would imply that a website can check whether a user has an HDR-capable screen and GPU.

Media playback depends on several factors. Besides the amount of processing cores and their speed, availability of hardware acceleration is taken into consideration [41]. But, it could depend on other system factors, such as the availability of external frameworks and libraries [4].

The main fingerprinting potential of this vector is to find the maximum allowed values for certain codecs. While we could not find any research, the bugtrackers of both Mozilla Firefox and Google Chrome indicated concerns about this fingerprinting vector. We can even extend this to make informed guesses to the hardware that is available in the system.

Of course, this means that repeated calls to these functions could expose which hardware is present in a given system. We can find out whether certain hardware acceleration would be available and which compile flags were enabled when compiling the browsers.

### 4.14.1 Experiment

We take the most popular video codecs, resolutions and framerates. For each combination, we probe to find the maximal bitrates to be allowed to play at all and to play 'smoothly'.

Our formats include x264, HEVC/x265, VP8, VP9 and AV1. We use the Matroska container format for x264 and HEVC/x265 video. For VP8 and VP9, we used the WebM container format. While AV1 is a relative new standard, it is quickly gaining attention as Google adopts it on their YouTube platform. The main advantage from a fingerprinting perspective is that there isn't a lot of hardware accelerated decoding hardware available. As of writing, only the latest generation of graphics cards from Intel, NVIDIA and AMD support hardware accelerated playback for AV1 video.

To pick our resolution, we pick several common 16:9 resolutions: 480p, 720p, 1080p, 1440p, 2160p and 4320p. As times goes on, these would have to be adjusted. We

look for video bitrates specifically, as video is more computationally expensive than audio to playback.

As video is often shot in different framerates, we do take the most popular ones into account as well. These are 15Hz, 23Hz, 24Hz, 25Hz, 30Hz, 50 and 60Hz.

For every combination of the previously mentioned values, we get a 3-tuple, containing the maximal bitrates that return playable, smooth and efficient playback. We use null if no such value could be found.

Our experiment found that our devices would say that the media was able to playback, no matter the bitrate. We initially tried to keep increasing the bitrate until a non-perfect result was returned. But, as this kept increasing, we capped the bitrate at a rate at which a raw video representation would be more efficient.

Enabling resist fingerprinting did Firefox not disable the API, which isn't in line with expectations (as resist fingerprinting should disable any API giving access to underlying hardware).

#### 4.14.2 Mitigation

Firefox thought about this fingerprinting potential in 2018. While the easy solution would have been to standardize return values, this would hurt user experience and make the entire API useless. A middle-ground was found in that Firefox now accurately returns whether media can be played back, but (if it can play) Firefox will always report that the playback should be smooth and power efficient [4].

As of writing, Chromium did not (yet) mitigate any potential fingerprinting. In August 2020, the developers launched telemetry collection to determine the entropy of the API between their user base [27]. After these results come in, they might decide on potential changes to the browser.

#### 4.14.3 Conclusion

While the Media Capabilities API exposes some information about the environment of the user, possibly its available hardware accelerated video playback and available configuration, the data that is exposed by this API doesn't harm the user. All data that we could extract from this vector, was already available using other APIs, such as the WebGL API.

This doesn't mean that the fingerprinting is harmless. In applications where it is vital to be anonymous, such as the Tor Browser, this API should be disabled. This is especially important when the browser would disable all these other APIs.

### 4.15 WebRTC

WebRTC is a protocol to allow real-time communication between hosts. The main protocol works as follows. First, both hosts have to discover how they are connected to the Internet. This is done using Interactive Connectivity Establishment (ICE). Here, browsers send all the IP addresses associated with this device to the server. This is done to discover hosts that might be on the same subnet and thus don't need any additional configuration.

The fingerprinting surface of this API comes in the form of IP addresses. While it is possible to identify IP addresses (see Chapter 5), an additional element here is that we can also have a list of all private IP4 addresses that a host has [59]. Additional information that can be gained, is in the form.

As this vector is from 2011, many browsers have fixed the issue and now only send public facing IP addresses [59].

## 4.16 Other components

Other components in a system, such as DRAM memory and a power supply, might have a more generic use. They do not have a (major) influence on performance.

### 4.16.1 Main memory

Take main memory (DRAM) for example. Its purpose is rather straightforward. They only talk with the memory controller. But, different memory speeds exist. Does this mean that fingerprinting is possible?

Especially with newer standards such as WebAssembly<sup>6</sup> on the rise, which will be less dependent on browser-specific implementations, we might see viable fingerprinting techniques in the near future.

That doesn't mean that this will remain the case. For example, a fingerprint reader (or any biometric sensor), might be exposed in the future using an API. And the current trajectory of the web implies that this will only be a matter of time.

## 4.17 Conclusion

As seen in this chapter, a lot of fingerprinting vectors are available on the API level. A table with an assessment for our criteria can be seen in Table 4.1. While individual vectors might not be problematic (with respect to privacy), combining them could easily make computers uniquely identifiable, in a way that would make it difficult to change the fingerprint of a configuration.

The main issue with APIs is their ease of use. Whereas traditional native applications would have to be downloaded and ran by a user before getting access to the hardware, these APIs provide the fingerprintable information to every website that a user visits. With traditional applications, users would probably know all the programs that they run - not to mention that there would probably be a AV solution to detect and stop the execution of potential malware.

Mitigations for many of these vectors could yield significant reductions in the available interactivity on the Web.

---

<sup>6</sup>WebAssembly, abbreviated as Wasm, is a new binary format alternative to JavaScript. It allows code written in system-programming and compiled languages such as Rust and C++ to be compiled into a format that is recognized by browsers.

Metric / Vector	Canvas2D font	WebGL Vendor	WebGL Shader	WebGL pixels	WebGPU		
Degree of entropy	Medium <sup>a</sup>	Medium	Low	High	Low <sup>b</sup>		
Amount of extracted data	amount	Medium	Low	High	Low		
Impact of complete Mitigation	Functionality	None <sup>c</sup>	None <sup>d</sup>	Functionality <sup>e</sup>	Functionality/Performance <sup>f</sup>		
Speed of execution	Fast	Instant	Instant	Fast	Instant		
Impact of execution	None	None	None	None	None		
Percentage of users vulnerable	Max. 96% <sup>g</sup>	Max. 97.6% <sup>g</sup>	Max. 97.6% <sup>g</sup>	Max. 97.6% <sup>g</sup>	<1% <sup>h</sup>		
Metric / Vector	WebAudio	Perf. API	Batt. API	Screen	VR/XR	Gamepad	Media cap.
Degree of entropy	Low	High	High	Medium	Low	Low	Unknown[27]
Amount of extracted data	Low	Low	Low	Medium	Low	Low	Low
Stability of the result	Browser	Hardware	Hardware	Hardware	Hardware	Hardware	Hardware
Impact of complete Mitigation	Functionality	Performance	Functionality	Functionality	Functionality	None <sup>i</sup>	Functionality
Speed of execution	Fast	Slow	Slow	Instant	Instant	Slow <sup>j</sup>	Fast
Impact of execution	None	Big	None	None	None	None	Some
Percentage of users vulnerable	81.21%	95.9%	75%	96.8%	70.9%	95.7%	91.4%

**Table 4.1:** Comparison of different fingerprinting vectors related to Web APIs<sup>a</sup>High on desktop, Low on mobile devices<sup>b</sup>Considering vectors that were newly introduced by WebGPU. High in other cases.<sup>c</sup>Normally, as WebGL is a standard, it should be vendor agnostic. That said, it could be that performance or results differ between vendors and optimization can be made.<sup>d</sup>Debugging shaders could still be enabled through settings for developers.<sup>e</sup>Several legitimate use cases exist to get the frame buffer.<sup>f</sup>If limits were to be set on what WebGPU can do, and those limits cannot be changed, there's an upper limit to the amount of graphics that can be drawn.<sup>g</sup>It's unclear how many users have some form of fingerprinting resistance turned on. If they did, all canvas fingerprinting vectors would not work. We give the maximum, which would be true if no-one is using resisting technologies.<sup>h</sup>WebGPU is still in active development and not yet mature<sup>i</sup>If the gamepad isn't touched, no data is leaked. Otherwise 'Functionality'<sup>j</sup>Extracting the gamepad availability is instant, but drift fingerprinting is slow.

## Chapter 5

# Fingerprinting the Network

Up to this point, most of our focus has been on data that could be extracted from either the browser (and accompanying system) or the application layer of the networking stack. In this chapter, we look at techniques that use properties found in lower layers of the networking stack. As most servers the user connects to aren't connected directly on the same network as the user, the main fingerprinting potential can be achieved on the networking layer.

A lot these techniques are also applicable in network fingerprinting and user fingerprinting, as discussed in 1.6. In this thesis, we will examine them only briefly, as it isn't the main topic of our research. But, the main difference with a more general network fingerprinting, is that we are at one end of the connection. This allows a website to have more control over the connection than someone outside trying to determine which website is being viewed by the user (or which user is connecting to a particular website).

### 5.1 IP address

When a user browses to a website, their IP address is sent to that website as part of the Internet Protocol. This is required for the website to be able to respond to the request. Depending on the type of network to which a user is connected, the type of IP address can be different. The device can have a global address, which means it's uniquely routable from the Internet.

While global addresses are decreasingly common in IPv4, they are common in IPv6. As the address space allows for  $2^{64}$  addresses on a single network (of which there are a maximum of  $2^{64}$ , routers will be able to provide a unique global address to each user, as long as fewer than  $2^{64}$  devices are connected at each given time. Also, ISPs are more likely to give static IPv6 prefixes than static IPv4 addresses, which increases the fingerprinting effectiveness.

Another scenario is that the user is browsing from behind a NAT<sup>1</sup>. In this case, depending on the NAT, there might be multiple users behind one IP address. In Belgium broadband connections, a NAT is generally used at customer-level, meaning one IPv4 address corresponds to one customer. In other countries, NAT deployments can include hundreds of clients in so-called CGNAT (Carrier-grade NATs). Mobile networks tend to use CGNAT as well.

---

<sup>1</sup>Network Address Translation, which allows users to have one global IP address and all have their own private IP address

In the case of a NAT, the IP address might still be feasible as a fingerprinting vector. The effectiveness depends on the amount of traffic and the diversity of the traffic, as websites targeting population in a certain country will have the IPs of ISPs in that country. If, however, a website targets a global audience, the entropy could grow to the entire address space (which is up to  $2^{32}$  in the case of IPv4). Another fingerprinting vector related to NATs, is obtaining the private IP address used by the client [75]. This can be done using the vector described in 4.15.

Mitigation of IP address tracking can be both easy and difficult. Public VPN companies offer to reroute traffic through their servers, using one IP for many clients. This mitigation comes with certain disadvantages. First, if only one user of a particular VPN server is accessing a website, it might still be possible to bring this information back. Also, these companies tend not to offer IPv6, which makes reaching IPv6 websites impossible. To introduce IPv6 without the increased risk of fingerprinting, would require an IPv6 NAT. Finally, the quality of the connection for users might be degraded. Besides the latency added by rerouting traffic through an external provider, companies such as Cloudflare practice more fingerprinting techniques on traffic from VPN servers, as attacks often originate from here [16].

Another approach is the The Onion Router, known as TOR. It provides a decentralized network to browse the Internet by relaying requests and responses over a global and decentralized network of Tor nodes. Each node adds encryption, so that only the last node in the chain can decrypt the request. During 2020, the Tor Project has implemented IPv6 support in the latest versions of their clients [9]. Disadvantages to using the Tor network exist. As traffic is routed through multiple nodes before a request is made, the added latency is bigger than a public VPN approach. A second disadvantage is that, due to the anonymity of the network, a lot of malicious traffic is routed through it [55]. This traffic will result in the IP addressess of exit nodes to receive captchas, much like VPNs [55].

## 5.2 Autonomous Systems

Tracking a user solely on IP address can infer many disadvantages, such as the change in IP address that can happen when a connection doesn't have a static IP address, a user might change ISP or go to another network (e.g. a smartphone changing from home broadband to mobile broadband). An improvement would be to use a broader description of information of this IP address, such as the block of IPs that were assigned to the ISP or the number of the autonomous system on which the IP address resides. Users are less likely to change their ISP than they are to receive a new IP address.

ICANN allows  $2^{16} = 65536$  different autonomous systems. To the best of our knowledge, no numbers exist that examine the distribution of users across these ASes, so we cannot tell how distributed requests will be. We can however say that websites that mainly target one specific region will have less benefits when using this as fingerprinting technique, as they'll only see the limited number of ASes that the ISPs of their target region make out.

We could extract this information more broadly to use the WHOIS information related to the IP address. This would give a general idea as to where the user is located. It would solve the problem with different networks, as we only use the provided information to gain a broad idea of the physical location of the user.



In Belgium, the granularity of this data doesn't seem to be specific. In our tests, we found that the WHOIS data associated with our IP addresses was often significantly different from our actual location. The data seems to be set to the nearest province capital, which provides a compromise between the users' privacy and the functionality of the WHOIS database.

Mitigations against this fingerprinting vector are similar to the IP fingerprinting, with the added difficulty that public VPNs also have to provide WHOIS information to the database in order to use their IPs.

## 5.3 IP headers

Our focus has been solely on the IP address and the related information thereof. But, other headers exist in the Internet Protocol.

While a lot of headers are also readable in network fingerprinting [80], some extra headers can be used in the context of browser fingerprinting, as a website administrator is at the end of the communication.

The main header is the TTL header. The TTL (Time-To-Live) ensures that packets don't remain in the network in an infinite loop. This could happen if devices have incorrect routing tables or a circle structure exists in the network. Another fingerprintable value about the TTL is that, unlike the Hop Limit value in IPv6, the initial TTL value of routers are vendor-specific and mostly a power of 2 [80]. This means that it is possible to fingerprint the type of router used by the client and how many hops away the user is located [80].

In our experiment, we recorded the TTL IPv4 and the Hop Limit in IPv6. We saw that these values remained stable over the course of the experiment, indicating that the network between our connection didn't change in the two weeks between runs of our experiment.

## 5.4 IPv6-specific

As IPv6 is still being integrated into existing networks, the availability of IPv6 itself can be a valid fingerprinting vector. If IPv6 is available, we can try to see which deployment of the new IP stack is available. One implementation is by natively supporting it, often in a dual-stack configuration with support for IPv4 as well. Others include an IPv6 tunnel, where traffic is sent over a IPv4 network to an IPv6-capable router that then makes the request on the IPv6 network. Finally, we can try to see which stack is preferred by the browser.

Recent Google statistics show that between 30% and 35% of their users request the search engine over IPv6 [34]. Most of the IPv6 deployments use a native connection, with the amount of tunneling decreasing [34].

To practically use these vectors, we can access both a resource on a domain which only has a IPv4 address and another domain that only resolves to an IPv6. By doing this, we can extract the two extra bits of data: the availability of IPv6 and whether it is preferred by the browser. Also, we are able to link the IPv6 address to an IPv4 if the user is in a dual-stack network. This can increase the fingerprinting potential in a case where the other stack is used or one wouldn't be available at a given time.



Another fingerprinting vector regarding IPv6 can be found in the SLAAC extension. This extension on the IPv6 standard allows devices to setup their own IP address on a new network, with the need for services such as DHCP [33]. It uses the router's IP address to determine the subnet of the network. It then generates an IP address based on its MAC address. As the MAC address of a device is unique, the entropy becomes as high that each device connected to a network generates a unique value. As such, any router that processes our packets can determine our MAC address and link it to the network on which the device is operating. Servers can use this to determine which device travelled from one network to another, even if the network prefix changes. As this is a privacy concern, a newer RFC resolves this by suggesting to use randomly generated addresses that aren't renewed [46].

Mitigation against this is possible by network admins (by disabling SLAAC). But, users of networks with SLAAC enabled will still require an IP address. To mitigate the vector from a user's perspective, MAC spoofing<sup>2</sup> can be applied.

## 5.5 TCP

Up to now, our focus on the network has been on the actual network layer. However, we've seen several exploits that were available on higher levels. While application layer vectors were discussed in Chapter 3, we didn't search for exploits on the transport layer.

In 3.1.3 we saw that the HTTP protocol allows to find out which port numbers would be available on a given network, which could lead to a portscan and find out possible firewall rules.

This, however, isn't the only fingerprintable component of the transport layer, and TCP mainly in the context of HTTP<sup>3</sup>.

The main fingerprintable element that we could find, was the congestion control that was used by TCP connections. Different operating systems (or versions thereof) can have different default configurations for the operating system.

## 5.6 Conclusion

While we only briefly explored the network layer browser fingerprinting, we could find some possible fingerprinting vectors. As changing the public IP address of a connection can have disadvantages, such as increased latency, the vector is rather difficult to mitigate without hurting user experiences.

A summary can be seen in Table 5.1.

---

<sup>2</sup>Randomizing the announced MAC address, not using the hardware provided address

<sup>3</sup>As HTTP/3 is in active development and uses userspace congestion control, we omit it for this comparison

Metric / Vector	IPv4 address	TTL	IPv6 address	IPv6 SLAAC	Preferred version of IP	AS	TCP congestion
Degree of entropy	High	Low	High	High	Low	Low	Low
Amount of extracted data	2 <sup>32</sup> bits	6 bits	up to 2 <sup>64</sup>	up to 2 <sup>48</sup>	1	2 <sup>16</sup>	5 bits
Stability of the result	Network	Network	Network	Device Network	Device Network	Network	Device Network
Impact of complete Mitigation	Performance Convenience	N/A	N/A	None	N/A	N/A	Performance <sup>a</sup>
Speed of execution	Instant	Instant	Instant	Instant	Instant	Fast	Slow <sup>b</sup>
Impact of execution	None	None	None	None	None	None	Performance
Percentage of users vulnerable	100%	100%	36% [34]	Max. 36% <sup>c</sup>	Max 36% <sup>d</sup>	100%	100%

**Table 5.1:** Comparison of different network-based fingerprinting vectors

<sup>a</sup>If all operating systems were to use the same congestion control algorithm, future performance and innovation will be impossible and thus hurt performance. As a lot of devices don't support ECN, it is likely to be disabled if all TCP had to be the same

<sup>b</sup>Requires a file transfer.

<sup>c</sup>We could not find any numbers from a reliable source about the specific extensions. We limit the amount of users by the available IPv6, based on [34].

<sup>d</sup>Assuming all users with IPv6 can have a dual-stack setup.



## Chapter 6

# Fingerprinting prevention

Up to now, we've discussed a variety of fingerprinting vectors that can be utilized to create a fingerprint. For each method, we've evaluated which mitigations techniques were possible to omit or reduce the effectiveness of the fingerprinting. New ways to gain more new data are discovered frequently, especially with the increasing amounts of new functionalities and added complexity in browsers. Therefore, for the foreseeable future, there will be an arms race between parties who discover new fingerprinting techniques and those who try to defeat it.

We've seen several mitigations that work on the same principles, such as asking for permissions and disabling some functionality. Can we find more broad approaches to mitigate fingerprinting, which could also work for fingerprinting vectors that are yet unknown?

In this section, we look at some general principles on mitigating fingerprinting and the reasons why these methods aren't applied more broadly.

### 6.1 Incentives to block fingerprinting

While there are definitely legitimate use cases for fingerprinting (as discussed in [1.4](#)), there are reasons to resist against the practices.

A first reason to resist against fingerprinting, would be the privacy implications and lack of control in the process. Cookies can be removed, a hardware based fingerprint cannot. Users might prefer not to be followed around the web, or face (legal) consequences if they were to be followed or identified. In some countries, such as China, these legal consequences can result in human rights violations.

As said before (in [1.4](#)), an incentive to commit to browser fingerprinting is to ensure session cookies aren't stolen. People with malicious intent are aware of this and try to steal more than just the cookie of their targeted users, but some aspects of a fingerprint [\[20\]](#). Recent research shows that websites exist to trade in fingerprinted data of users [\[20\]](#). This fingerprint data is collected by various means, e.g. pay users a small reward to install a given program or pay website owners to collect these values [\[20\]](#). When a user were to resist fingerprinting, less data of this kind would be available.

## 6.2 Disabling JavaScript

A popular solution to reduce the amount of fingerprinting that can be done on the web, would be to disable the execution of JavaScript code. A lot of the browser fingerprinting vectors we've discussed rely on client-side scripting to work.

While this would reduce the fingerprinting surface, it would give several disadvantages.

The first disadvantage is that JavaScript is ubiquitous. Progressive web apps, which are becoming more popular, rely on client-side scripting. If users were to disable JavaScript entirely, they'd not be able to use those services. One solution would be to allow JavaScript on a per-site basis, which would require third-party plugins such as NoScript. As this wouldn't eliminate the fingerprinting potential for these websites.

The second one is that the lack of JavaScript support is a fingerprintable fact. Websites can add a reference to an image in the `<noscript>`-tags of browsers and then analyze. Or, they can do the opposite and make a particular request in JavaScript. If scripts were to be blocked only partially, we can apply the same techniques as are used to detect adblockers and extensions.

Finally, we've shown several approaches that don't require JavaScript to fingerprint a setup.

## 6.3 Browser Built-in Solutions

While we could put all responsibility to mitigate fingerprinting at the end user, browsers themselves can offer features to ensure that users are more difficult to fingerprint.

### 6.3.1 Chromium-based

To the best of our knowledge, Chromium doesn't provide any additional settings that can be configured to enhance protection against fingerprinting, in the way that other browsers do. The developers have stated their intentions to mitigate fingerprinting vectors with a new Privacy Sandbox initiative, which tries to mitigate cross-site tracking without hurting legitimate use cases [56]. Besides this, Chrome developers are working on more advanced fingerprinting protection to include in the browser in the future. An example thereof can be seen in 6.4.

Microsoft Edge, also based on Chromium, offers more tracking protection. Users can choose between three levels, where the default value is 'Balanced'. According to Microsoft, all levels provide protection against fingerprinters. However, we found a lot of vectors to be still working (such as the WebGL pixel buffer in 4.5), regardless of the level of protection we chose.

The Brave Browser is another Chromium-based browser. It advertises with included privacy protection and ad blocking. The included privacy protection mainly originates from the inclusion of the Privacy Badger extension as part of the browser. When using the strict option, APIs known to be used in fingerprinting, are given random values [69].

### 6.3.2 Firefox

Firefox advertises with the slogan of being ‘the privacy browser’. In recent updates, several general mitigations were introduced to further resolve the fingerprinting surface [24]. As discussed in 3.1.4, it now stores the cache on a per-site basis.

The browser offers different kinds of protection against fingerprinting. By default, the enhanced tracking protection is enabled. This will block known fingerprinting scripts [5]. These settings can be disabled on a per-domain basis, as some websites rely on third-party cookies for functionality. The lowest level. To completely disable the tracking protection, the user has to choose for the custom option and uncheck all options.

Besides enhanced tracking protection, Firefox offers more thorough tracking prevention as well. This setting is located in the about:config page, where settings that are oriented more towards advanced users can be changed. Among others, this setting disables the option to extract pixel data from WebGL, access to the Media Devices and specific fonts from the PC aren’t available [5].

‘resistfingerprint’ isn’t enabled by default (or taken into the main settings interface) because it will break APIs and other functionality. Users who don’t understand the setting or its implications will think the browser is broken.

### 6.3.3 Tor Browser

The Tor Browser is a browser used by people who wish to remain anonymous on the Internet. Among its users are whistleblowers and decedents of oppressive regimes who could face human rights violations if their identity were compromised. As there is a possibility for browser fingerprinting to be used to link a user to a connection, the browser tries to eliminate all fingerprinting options.

As the goal of the Tor Browser is absolute anonymity, many compromises are made in terms of user experience. The browser has the most aggressive form of fingerprinting protection among the browsers we’ve tested. By default, the previously discussed flag ‘resistfingerprinting’ (used in Firefox) has been enabled in this browser. Also, the NoScript extension is installed and enabled by default, which allows for individual scripts to be whitelisted and only on particular websites.

In vectors where such mitigations aren’t possible, the Tor Browser tries to be as generic as possible. To mitigate the header-based fingerprinting, it will e.g. use the most popular browser’s browser name and version, in combination with the most popular Accept-Language values.

The Onion network, on which the browser resides, has been configured to regularly change the exit node. This approach makes it that the network based fingerprinting methods are infeasible, as the exit node used will change over time. Network fingerprinting could still be possible if the attacker controls a network close to the user by checking connections to known Tor entry nodes.

### 6.3.4 Safari

The Safari Browser, which is made by Apple, is the default browser of MacOS and iOS. It is based on the Webkit browser engine, which is where the Blink codebase (used in Chromium) is forked from. Webkit has a more conservative vision on API

implementation. For the few vectors that we discussed, we saw that Safari often lacked an implementation, because developers didn't see an advantage in implementing it. Instead of offering their users all possible APIs. This means that these fingerprint vectors, such as the Battery API (4.9), isn't available in Safari.

Another method used by Apple is the stricter control they keep over the available plugins. On other platforms, developers are free to create plugins. Apple manually approves every plugin that is available for the Safari browser. This does make the browser more vulnerable to extension fingerprinting.

## 6.4 Differential Privacy and Privacy Budget

Another pattern that was alluded in mitigations of multiple APIs (e.g. 4.5 and 4.9) was to make the information gained from these APIs less accurate and include some form of noise to the output of the high-entropy API calls.

When looking at tracking protection, we can see a similar pattern. Privacy Badger is a browser extension that tries to mitigate third party cookie tracking, without blocking all cookies and without any block list. It relies on heuristic approaches, where it looks at which cookies are recurring and could thus have the potential to track users. If the extension sees that a website is used as third party by several other websites, cookies won't be stored. When a website gains more traction, connections to the (sub)domain are blocked. If a users experience a website is not working, they can re-enable each website individually (or disable the extension for this domain).

A similar technique to find out whether fingerprinting has taken place, was effective [25]. But, we now search for a technique where we wouldn't have to block the APIs.

One concept that can be applied in this context, is the notion of *differential privacy*. Here, repeated calls to identifiable information should include some entropy. As we said before, it could be possible to retrieve the distribution of this entropy by making multiple calls to the given attack vector. To mitigate this, differential privacy states that more noise should be added, based on the previously given noise, and this for every subsequent call.

Google introduced the concept of a *Privacy Budget* [39]. The concept is to count calls to known-fingerprintable functionality and see whether they exceed a given threshold. After this threshold is exceeded, the client will answer to requests with data that is less fingerprintable. This can be either by using differential privacy, making sure subsequent calls give the same output or indicating to a website that it's budget is consumed.

Some applications, such as online applications that rely on fingerprintable APIs, might need more calls than allowed by the Privacy Budget [39]. The concept should thus involve a way for users to allow a website to exceed the given budget. An example of such application would be an online photo editor that uses WebGL for hardware accelerated calculations, which would need canvas data to export the picture.

As of the moment of writing, the project is still in the concept phase. The developers wait for the results of entropy studies conducted by Google Chrome, where they collect data on how fingerprintable different APIs are [27]. This information is useful to find out which weights should be allocated to different fingerprinting APIs.

## 6.5 Conclusion

While it is possible for users to reduce the amount of fingerprinting that a website can do, it seems impossible to guarantee complete security. Even if all fingerprinting vectors in the browser and operating systems were to be resolved, website owners can still exploit the fingerprinting vectors on the lower levels of the networking stack.

Also, trying to mitigate fingerprinting often leads to an increase in the fingerprinting potential, as only a small amount of users will take the extra steps required to mitigate fingerprinting.

We saw several browsers take different approaches to mitigate the most prominent fingerprinting vectors. With the exception of the Tor browser, all major browsers seem to make some trade-off between privacy and usability, with some offering users choices.

So, mitigating fingerprinting is only effective if a significant amount of users of the fingerprinting website were to enable it. Otherwise, users who actively mitigate fingerprinting will be more fingerprintable. If Chromium-based browsers were to use the Privacy Budget, mitigation might be more feasible.





## Chapter 7

# Gaming as additional fingerprinting vector

Now that we have a broad view on the landscape of browser fingerprinting in the browser, we can see which techniques would be useful to utilize in a game context.

While including a game for fingerprinting purposes might give the impression of being a far-fetched idea, we are able to provide some reasons where the idea might have benefits.

First of all, many corporate website already include graphical intensive animations (and sometimes little games). Adding a game here wouldn't be considered suspicious in general.

Moreover, if we can prove that a reliable way exists to fingerprint using a game element, new Captcha-like technologies could be made more enjoyable for users.

Coming back to our mitigations, we saw the new Privacy Budget initiative by Google (6.4). If browsers were to put limits on the amount of resources allowed to fingerprint, a game is a valid reason to ask for an exception to exceed the budget.

### 7.1 Selection of Techniques

While a lot of techniques can be used, we will focus our attention on the techniques that are of direct relevance in a game.

To show a tutorial for our game, we can use the Media Capabilities API to find out which codecs can work on our given machine. This gives us a legitimate reason to use the API and utilize its fingerprinting surface. We can even use the screen dimensions to serve a better video.

As most prominent fingerprinting vector that we could find, we use our new techniques concerning the Gamepad API. We can use it to fingerprint the amount of drift in both axes. We can use AI to detect whether a user made the movement that we asked for. Mobile devices, such as smartphones, might not have a gamepad at hand. To compensate for the loss in data, we can use the accelerometer and gyrosensor to fingerprint the likelihood of the user being a legitimate user. If done correctly, we could try to measure the differences in orientation when a user touches the device (e.g. to make a keypress) to ensure that the event isn't virtually created.

We include the WebXR standard to detect whether any virtual or extended reality devices are connected. We don't use them actively, as using a virtual headset uses

different controllers that don't have the same analog sticks. It wouldn't be strange for a game to check the availability of XR devices.

The Performance API can be used to benchmark the CPU of the device. While this doesn't immediately add anything to the game, people are less suspicious of a sudden surge in CPU usage when a game is running. The length of the game can be correlated to the length required for the benchmark to run.

We don't consider benchmarking the GPU in a similar way to the CPU. The reasons are that we can know the manufacturer and model by calling the WebGL debugging shaders. We can use those values to compare to a database. This would not only yield better performance, it could shorten the fingerprinting period. We do want to say that this kind of benchmarking would be possible, e.g. by showing a scene and then increasing its complexity until the framerate of our game is below a given threshold.

The exclusion of GPU benchmarking doesn't mean that we cannot use the GPU at all. We will still perform the better-known WebGL Canvas Fingerprinting. At a given page, we will ask for frame data. If we introduce a purpose for the fingerprinting, we can even try to work around the resist fingerprinting settings.

Finally, resources made available by the game can be stored in the browser cache, depending on the identifier that we give them. To extend our identifier, we need to split the code for our game into more separate files.

## 7.2 Analysis

As the techniques discussed concerning the Media Capabilities API are depending on the amount of entropy between devices, we lack real world numbers. We do expect to see similar results between devices.

When using the two sticks that are typically found on a gamepad, each with two axes, we can gain 4 new fingerprintable datapoints. Considering that we subdivide them into groups (giving 2 bits of data), we gain 8 bits of fingerprinting data with this vector. While no numbers exist (to our knowledge) about the availability of gamepads (as browsers hide them), we consider it safe to assume that only a minority of the users will have a controller at hand. As we only have a fallback for mobile devices (or devices with accelerometer), further work should focus on other techniques to gain more entropy.

## 7.3 Further work

Due to time constraints, we were not able to actually implement a proof of concept of this fingerprinting game. Further research could be done to show whether the techniques explained are effective in collecting more fingerprinting data in practice.

## Chapter 8

# Conclusion

The domain of browser fingerprinting is vaster than anticipated when starting the thesis. While trying to provide a survey, we found that the domain was too broad for a complete and in-depth analysis. To pick our fingerprinting vectors for analysis, we chose the available vectors in respect to their use in a game.

While it is well-documented how fingerprinting can be done in a multitude of vectors, little is known about the distribution of these values among Internet users. Numbers in this regard are often collected by privacy activists, whose audience cannot be seen as a representation for all users.

As the practice of fingerprinting is well-integrated, it can be difficult to mitigate properly without having a negative impact on the experience of users. Browsers are taking steps to reduce fingerprinting, and with initiatives such as Google's Privacy Budget, practices can become less obvious and intrusive. At the current state of the Internet full mitigation is impossible. Users cannot know with certainty that a visited website doesn't store their IP address or performs any fingerprinting on the network layer.

In regard to our game, we were able to find a combination of techniques that would allow to generate a reliable and difficult to mitigate fingerprint. While we were able to find new fingerprinting data using a game, this data wouldn't impact a typical user. The data requires special hardware to be available and used in order to use it for fingerprinting. Therefore, we don't deem the increase in fingerprinting surface significant.

However, in the course of our research, we found other reasons for the a game object as fingerprinting method to exist. As browsers try to restrict a website's ability to fingerprint, e.g. by measuring the amount of fingerprint-likely calls that are made, a game can be a valid reason to require an exception to this rule.

As these mitigation techniques are mostly conceptual at the time of writing, further research should be conducted when it materializes.



## Appendix A

# Dutch summary - Nederlandse samenvatting

*Due to university requirements, a Dutch summary is included as appendix to this thesis.*

Het doel van deze thesis is tweeledig. Enerzijds proberen we een oplistings te maken van veelgebruikte en historisch belangrijke fingerprinting vectors. Anderzijds kijken we naar hoe we nieuwe technieken kunnen aanwenden om van een game gebruik te maken om een meer volwaardige fingerprinting vector te maken.

## A.1 Achtergrondkennis

Het HTTP protocol biedt weinig state management. Doordat het oorspronkelijk doel enkel intern wetenschappelijk gebruik was, was er geen nood aan. Om state toe te voegen, is het concept van cookies geïntroduceerd. In essentie is een cookie een key-value pair dat een browser als HTTP header meestuurt bij elk verzoek aan een website.

Een groot voordeel is hun relatief eenvoudige setup. Een client moet enkel een string bijhouden per domein en servers kunnen de HTTP header uitlezen. Cookies bieden echter enkele nadelen. Zo zijn ze eenvoudig door gebruikers te veranderen, wat maakt dat ze steeds geverifieerd moeten worden. Ook bestaan verschillende aanvallen om cookies te stelen van gebruikers, zoals diverse cross-site scriptingaanvallen. Één van de oplossingen hiervoor kan gevonden worden in browser fingerprinting.

Browser fingerprinting is het onderzoeksdomein dat zoekt naar verschillen tussen bezoekers van websites. Deze verschillen kunnen dan gecombineerd worden om een soort vingerafdruk te maken. Net als een menselijke vingerafdruk, laat een browser hem vaak ongewild achter bij bezoek en is het moeilijk te detecteren of een website er gebruik van maakt. Ook is het niet eenvoudig om bepaalde aspecten aan te passen.

Vanuit het perspectief van een website beheerder, zijn er verschillende redenen voor websites om aan fingerprinting te doen. Zoals eerder gezegd, kan het als veiligheid dienen voor cookies. Daarnaast is het een nuttig element voor browsers om mensen te onderscheiden van bots. Het zoeken van bots, een verzamelnaam voor geautomatiseerde bezoekers, kan helpen bij spampreventie.

Naast spampreventie, is fraudebestrijding een andere reden voor fingerprinting. Zoals eerder gezegd is het mogelijk voor cookies om gestolen te worden (bv. door gebruik te maken van XSS zwakheden). Wanneer we cookies linken aan elementen van de fingerprint, is het mogelijks moeilijker voor aanvallers om gebruik te maken van deze cookies. In het geval dat dan een cookie niet overeenkomt met de fingerprint die we verwachtten, kunnen we een extra manier van authenticatie vragen (bv. een SMS bericht). Dit concept staat ook bekend als RBA (Risk-based Authentication).

## A.2 Opzet van het experiment

Fingerprinting vectoren zijn op verschillende manieren van elkaar te onderscheiden. We maken gebruik van verschillende criteria om de vectoren met elkaar te vergelijken. Deze criteria omvatten:

- Graad van entropie: hoe verdeeld zijn de uitkomsten?
- Hoeveelheid data: hoeveel nieuwe informatie kunnen we hieruit verkrijgen? We proberen dit te ratificeren in bits.
- Stabiliteit: welke wijzigingen zijn nodig om een andere uitkomst te verkrijgen?
- Impact van volledige mitigatie: indien we de vector onbruikbaar maken, welke impact zou dit hebben op de gebruikers?
- Snelheid van uitvoering: hoe lang duurt het om de vector uit te voeren?
- Impact van uitvoering: wat merkt de gebruiker van de uitvoering van de vector?
- Percentage van gebruikers dat kwetsbaar is

Om de experimenten uit te voeren in een eenvormige omgeving, hebben we een framework gemaakt dat ons toelaat om in een gestructureerde manier verschillende vectoren te vergelijken. Dit framework, genaamd 'Fingerprint Lab', maakt gebruik van een progressive Web App, in combinatie met een minimale backend om uitkomsten van vectoren te vergelijken met andere systemen waarop dezelfde testen gedaan werden.

De frontend maakt gebruik van Vue, Vuetify en TypeScript om een algemeen framework te hebben om onze experimenten parallel uit te kunnen voeren. Om verschillende vectoren te implementeren, maken we gebruik van een TypeScript interface, die gelijkaardig aan Java interfaces werken. De backend maakt gebruik van Cloudflare Workers en slaat gehashte waardes van de effectieve fingerprinting data uit ons framework op.

Het framework heeft ook enkele tekortkomingen. Zo houdt het op dit moment enkel hashwaardes bij van de fingerprinting vectoren, waarbij er geen correlatie is tussen de gelijkaardige waardes. Ook is het mogelijk om bij shallow waardes te achterhalen welke clients welke waarde hebben geproduceerd, wat dus de exacte configuratie van een systeem alsnog kan tonen.

## A.3 Browser

In de volgende hoofdstukken focussen we op verschillende fingerprinting vectoren op verschillende plaatsen in de omgeving waar het mogelijk is om fingerprinting gegevens te verkrijgen. In de eerste instantie kijken we naar de browser zelf en welke vectoren hier mogelijk zijn.

Een eerste en duidelijke manier om fingerprinting te doen, is door gebruik te maken van de HTTP headers voor User-Agent, Accept-Language en DNT (Do Not Track). Deze headers geven een eerste manier om een browser te fingerprinten. De hoeveelheid entropie is echter onvoldoende om een unieke waarde te verkrijgen (veel gebruikers blijven over per bucket).

Dit is niet de enige attack die in het HTTP protocol aanwezig is vanaf versie 1.1. De Alt-Svc header geeft aan op welke poorten een website nog geserved wordt. Door hiervan gebruik te maken, is het mogelijk om clients in te delen per poortnummer en IP-adres dat ze gebruiken. Ook kan veelvuldig gebruik hiervan leiden tot een poortscan op de firewall van het netwerk waarop de computer is aangesloten.

Wanneer we dit uitbreiden naar nieuwere iteraties van het HTTP/2 protocol, zien we onder andere dat ook de introductie van streams leidt tot een aantal standaardwaarden die fingerprintable zijn.

De nieuwste iteratie, HTTP/3, heeft als belangrijkste nieuwigheid op het gebied van fingerprinting een 0-RTT connection setup, waardoor deze de encryptieparameters in de browser kan laten opslaan. Doordat elke gebruiker een unieke encryptiekey heeft, is het zeer waarschijnlijk dat dit gebruikt kan worden voor fingerprinting. Deze fingerprinting aanpakken kan door geen gebruik te maken van de 0-RTT infrastructuur, wat tot een kleine extra latency leidt bij het opzetten van verbindingen.

Naast het HTTP protocol zijn andere elementen van een browser aan te vallen. We bekijken grondiger hoe de cache bijvoorbeeld kwetsbaar is. Aanvallen omtrent de cache zijn ook nuttig in correlatie met een game, omdat een game verschillende assets heeft die in de cache kunnen belanden. We bekijken recente ontdekkingen op het gebied van favicon caching, die tot op het moment van schrijven in verschillende configuraties nog mogelijk is. Alhoewel de favicon exploit niet heel nuttig is voor de game, is het wel nuttig om de techniek te extrapoleren naar andere attacks in de browser cache.

Verdere aanvallen, zoals browser extension probing, NPAPI extensions en font probing, bekijken we kort.

## A.4 Web API's

Naast elementen van de browser, omvat een browser ook APIs om websites toegang te geven tot lower-level functionaliteit. Omdat we uiteindelijk willen werken naar een game, ligt de meeste focus van ons werk op dit stuk. De meeste functionaliteit van een dergelijke game zou zich baseren op de beschikbare fingerprintbaarheid van de APIs.

Hierbij bekijken we eerst verschillende attacks in het HTML5 Canvas, een element dat general-purpose graphics aanbiedt. Per canvas kan een website gebruikmaken van verschillende backends om graphics te tonen op dit element.



Een eerste mogelijkheid is de 2D graphics context. Hiervan is reeds aangetoond dat het fingerprintbaar is. In onze testen bleek hier echter een groot verschil te zijn, omdat verschillende combinaties van browsers met operating systems dit niet berekenen met hardware acceleratie. Andere gebruiken dan weer een implementatie die gelijkaardig is aan diegene die ze gebruiken voor WebGL.

De andere mogelijkheid is WebGL, een standaard door de Khronos Group om met OpenGL accelerated graphics toe te laten op websites. Onze eerste gedachte was dat browsers deze WebGL calls aan de GPU driver geven, die ze m.b.v. de OpenGL API kan uitvoeren, maar dit bleek niet waar te zijn. Op veel platformen wordt gebruik gemaakt van de ANGLE-library om de calls om te zetten naar een andere API (op Windows zetten zowel Chrome als Firefox de calls om naar Direct3D, op Linux maakt Chrome nog steeds gebruik van ANGLE, maar de calls zijn wel degelijk nog in OpenGL). Naast het feit dat de WebGL renderer (cf: de GPU) kan worden uitgelezen, is het ook mogelijk om gebruik te maken van floating point rounding errors om een unieke fingerprint te krijgen. Ook kunnen gecompileerde shaders opgevraagd worden, waarin kleine verschillen zitten.

Tot slot is er WebGPU, die ook op een canvas kan tekenen. Deze standaard is nog in ontwikkeling, maar zal dezelfde floating point rounding-problemen hebben als WebGL vandaag de dag. Ook zijn er voor deze API limieten vastgelegd voor bv. memory use, maar browsers kunnen ervoor kiezen deze limieten anders te leggen. Wij kregen WebGPU enkel werkend op Chrome op Windows, waardoor we nog geen experimenten met deze standaard konden doen.

Naast grafische elementen, vinden we ook fingerprinting terug in de Web Audio API. We zien dat verschillen er tussen browsers bestaan omtrent reproductie van geluidsgolven, waaruit vectoren ontstaan, analoog met de pixel buffers van het HTML5 Canvas.

Ook de batterij is mogelijk te fingerprinten, al vraagt dit een grotere duur en is dit in alle grote browsers verholpen. Het was mogelijk om, door een te grote precisie in de API output, een idee te krijgen over de staat van een ontladende (of opladende) batterij.

In de context van de game, hebben we ook gekeken naar de WebXR API, waar we vooral zien dat het detecteren van een XR-apparaat (zowel een VR- als een AR-bril, als bijhorende controllers) mogelijk is, zonder dat de gebruiker een bevestiging dient te geven. Een vorige standaard, de WebVR standaard, liet dit echter niet toe.

Het verkrijgen van een scherm-resolutie en kleurenruimte is mogelijk. Behalve dat hierdoor websites een hardware element kunnen fingerprinten, is het ook mogelijk om hieruit af te leiden welke balken er open staan in een browser. Een game zou hier ongemerkt gebruik van kunnen maken, omdat die bv. moet weten hoe groot de applicatie is op een volledig scherm.

Meer aansluitend bij onze game, komen we eerst bij de Gamepad API. In verschillende werken wordt deze vector 'gewoon' uitgelezen en toegevoegd aan het fingerprinting surface, maar hierbij wordt relatief weinig uitleg gegeven over hoe dit concreet zou ingevuld worden, wellicht om enkel de beschikbaarheid van een gamepad te detecteren. Dit is echter vrij beperkt, aangezien we zien dat gamepads standaard niet beschikbaar gemaakt worden voor webpagina's (tenzij een gebruiker een knop indrukt of een as verplaatst). Als we echter in de context van onze game kijken naar de beschikbaarheid van een gamepad, kunnen we meer data verkrijgen. Naast het

type van controller, is het mogelijk om een idee te krijgen van het operating system en de browser (al is deze data reeds uit andere vectoren te verkrijgen). Wat we echter ook merkten, is dat verschillende browsers een andere implementatie hebben wat betreft het verkrijgen van de input. Doordat sommige browsers de exacte waarden doorsturen, is het mogelijk om de drift van controllers te weten te komen. Enkel Firefox op Windows bleek hiertegen bescherming te bieden (door het gebruik van de DirectX APIs die Windows beschikbaar stelt).

Ook is de Media Capabilities API beschikbaar. Deze API stelt ons in staat om te kijken welke media vlot, niet of mogelijk afgespeeld kunnen worden op een systeem, en of deze playback power-efficient is. Over de fingerprintbaarheid van deze API bestaat nog enige onzekerheid. In theorie zal deze API op verschillende systemen andere waarden aangeven, maar veel systemen hebben bv. hardware-accelerated playback van populaire cdec's, wat maakt dat veel output hetzelfde zou zijn. Chrome is op het moment van schrijven bezig met te verzamelen hoe groot de entropie is op deze API om te kijken of mitigaties nodig zijn.

Tot slot keken we kort naar de WebRTC standaard. Deze standaard laat verbindingen toe tussen hosts, eventueel via P2P (afhankelijk van het netwerk van de twee clients). Om te detecteren of zo'n verbinding nodig is, moet bekeken worden op welke manieren een host te bereiken is. Voorheen kon de setup-server (STUN-server) dan ook private IP-adressen van een host te weten komen. We zien dat de meest-gebruikte browsers hebben dit probleem inmiddels verholpen.

Voor de overige componenten van een systeem (zoals geheugen) zagen we niet onmiddellijk methodes om aan fingerprinting te doen.

## A.5 Network

Op lagere lagen van de stack is het ook mogelijk aan fingerprinting te doen. We beperken ons hier tot de basis omdat dit geen al te directe invloed heeft op onze game.

Een eerste element is het gebruik van het IP-adres. Al geeft dit een goede algemene indruk, door de verschillen in implementaties (NAT of CGNAT) is het mogelijk moeilijk om unieke verbindingen eruit af te leiden. Ook is het mogelijk om publieke VPNs te gebruiken om deze vectoren te mitigeren.

Naast dat IPv6 meer IP-adressen heeft, is er ook een bepaalde configuratie waarin de clients hun MAC-adres moeten gebruiken om stateless een IP-adres te configureren. Hoewel de IP-adressen dan verschillen, is het toch nog mogelijk om ze te linken doordat ze eindigen op het MAC-adres van het apparaat.

Omdat het mogelijk is om IPv6-adressen te veranderen, kan naar hogere niveaus gekeken worden. Zo kan het netwerk (bijvoorbeeld op /64-niveau) of het AS waarin het adres zich bevindt. Omdat dit meer hosts bevat, zal de entropie licht dalen, maar de stabiliteit verhogen. De kans is vrij klein dat een gebruiker van ISP verandert.

Uit al deze vectoren is het ook mogelijk om metadata te verkrijgen die hoort bij het IP- of AS-adres. Dit kan gebruikt worden om bijvoorbeeld het land of een schatting van de woonplaats te verkrijgen.

De transport layer is mogelijk ook fingerprintbaar. TCP-stacks kunnen verschillen in de manier waarop aan congestion control gedaan wordt. Zo kan een TCP-stack

ECN ondersteunen en een veelvoud aan beschikbare congestion control algoritmes gebruiken. Veel van deze gegevens zijn echter gelinkt aan het operating system dat gebruikt wordt, wat een fingerprinter al kan verkrijgen door de User-Agent string te bekijken.

## A.6 Mitigatie

Per techniek zagen we ook hoe we de specifieke technieken kunnen verhelpen. We zien hier echter een aantal patronen terugkeren. Het is daarom ook nuttig om te zoeken naar algemenere manieren om aan mitigatie te doen. Hierbij kijken we naar browser-specifieke technieken om fingerprinting tegen te gaan, maar evenzeer naar dingen waar makers van APIs rekening mee dienen te houden om de mogelijke fingerprinting te verminderen.

De W3C, die de meeste Web APIs maakt en standaardiseert, vraagt auteurs van nieuwe standaarden om te kijken welke fingerprinting mogelijk is. Dit maakt dat er bij nieuwe standaarden goed zicht is op de zwakheden en hoe browsers ermee om kunnen gaan om de impact te minimaliseren.

Een groot probleem met Web APIs specifiek is dat ze vaak een grote hoeveelheid data vrijgeven aan elke arbitraire website die een gebruiker bezoekt. Nochtans vertrouwt een gebruiker niet elke website evenveel. Beter is om meer permissies te vragen per pagina die de gebruiker bezoekt. Dit zou echter een groot nadeel geven voor gebruikers, aangezien veel websites niet gebouwd zijn hiervoor en dus code zou niet meer werken (door bv. excepties te geven zonder dat ze opgevangen worden) als gebruikers geen toestemming geven. Hierdoor gaan gebruikers mogelijks in de toekomst sneller en meer toestemmingen geven, wat het opzet ondermijnt.

Browsers bouwen ook andere beschermingen in. Zo heeft Firefox een ingebouwde bescherming tegen gekende trackingscripts en fingerprinters, die in een aantal gevallen ook al standaard aan staat (en anders eenvoudig te activeren is). Mocht hierdoor een website niet meer functioneren, dan is het mogelijk deze per site uit te zetten. Daarnaast kan met de optie ‘resist fingerprinting’ nog meer bescherming aangezet worden, weliswaar ten koste van functionaliteit.

Chrome heeft minder bescherming ingebouwd, maar is aan het werken aan een nieuw concept hieromtrent. Dit concept is het Privacy Budget, waarbij een browser zou bijhouden welke fingerprint-gevoelige acties een website uitvoert. Indien een threshold overschreden wordt, zal de browser zich dan harder verzetten tegen verdere fingerprinting.

De Tor browser is het meest resistent tegen fingerprinting van alle populaire browsers. Door de categorie van gebruikers kiest de browser ervoor om de privacy als belangrijkste factor te nemen, terwijl andere browsers eerder een afweging maken van de impact op gebruikers t.o.v. het beoogde doel. Ook bevat de browser alle beschermingen van Firefox.

## A.7 Gaming

We gebruiken tot slot de technieken die we in de voorgaande hoofdstukken besproken hebben voor een game. Het gebruik hiervan heeft meerdere voordelen. Enerzijds willen we meer fingerprintbare data halen uit APIs en technieken die door

games specifiek gebruikt worden. Anderzijds is deze techniek al toekomstzekerder (het Privacy Budget zal een uitzondering moeten aanbieden voor games, aangezien die moeten pollen naar controller status).

In deze game vragen we aan gebruikers om bepaalde bewegingen te maken met de analoge sticks van een controller. Door de gradatie van drift te lezen, kunnen meer fingerprintbare gegevens krijgen. Verder kunnen we de assets van onze game in de browser cache steken, waarbij we fingerprinting vectors uit de cache kunnen gebruiken.

## **A.8 Conclusie**

Door de breedte van het domein van browser fingerprinting, is het niet gelukt een volledig overzicht te verkrijgen van het domein en alle beschikbare vectoren te gebruiken en/of testen.

We toonden aan dat een game een meerwaarde kan bieden in het fingerprinten van gebruikers. Deze extra data komt niet ten koste van de privacy van gebruikers, omdat het vereist dat gebruikers een controller moeten gebruiken, of de data is reeds beschikbaar voor fingerprinting door manieren die hier geen gebruik van maken.

Doordat nieuwe beschermingen tegen fingerprinting op komst zijn, kan de game een uitweg bieden voor websites die nog steeds aan fingerprinting wensen te doen. Het biedt een rechtmatige uitzondering op het relatief nieuwe concept van het Privacy Budget.



# Bibliography

- [1] url: <https://www.hcaptcha.com/>.
- [2] url: <https://coveryourtracks.eff.org/>.
- [3] url: <https://addons.mozilla.org/en-US/firefox/search/?promoted=recommended&sort=users&type=extension>.
- [4] url: [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1461454](https://bugzilla.mozilla.org/show_bug.cgi?id=1461454).
- [5] In: *Firefox Support* (). url: <https://support.mozilla.org/en-US/kb/enhanced-tracking-protection-firefox-desktop>.
- [6] 2021. url: [https://en.wikipedia.org/wiki/Banner\\_grabbing](https://en.wikipedia.org/wiki/Banner_grabbing).
- [7] 2021. url: <https://w3ctag.github.io/security-questionnaire/>.
- [8] 2021. url: <https://www.w3.org/TR/gamepad/>.
- [9] gaba 14. *The State of IPv6 support on the Tor network*. 2021. url: <https://blog.torproject.org/state-of-ipv6-support-tor-network>.
- [10] Gunes Acar et al. “FPDetective: Dusting the Web for Fingerprinters”. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer I& Communications Security*. CCS ’13. Berlin, Germany: Association for Computing Machinery, 2013, pp. 1129–1140. isbn: 9781450324779. doi: [10.1145/2508859.2516674](https://doi.org/10.1145/2508859.2516674). url: <https://doi.org/10.1145/2508859.2516674>.
- [11] 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*. 2014, pp. 674–689.
- [12] *Adobe Flash Player: Security Vulnerabilities*. url: [https://www.cvedetails.com/vulnerability-list.php?vendor\\_id=53&product\\_id=6761&version\\_id=&page=1&hasexp=0&opdos=0&opoc=0&opov=0&opcsrf=0&opgpriv=0&opsqli=0&opxss=0&opdir=0&opmemc=0&ophttps=0&opbyp=0&opfileinc=0&opginf=0&cvssscoremin=0&cvssscoremax=0&year=0&month=0&cweid=0&order=3&trc=1078&sha=ac2a72f983d2b7488412b74b424af6da7078c21a](https://www.cvedetails.com/vulnerability-list.php?vendor_id=53&product_id=6761&version_id=&page=1&hasexp=0&opdos=0&opoc=0&opov=0&opcsrf=0&opgpriv=0&opsqli=0&opxss=0&opdir=0&opmemc=0&ophttps=0&opbyp=0&opfileinc=0&opginf=0&cvssscoremin=0&cvssscoremax=0&year=0&month=0&cweid=0&order=3&trc=1078&sha=ac2a72f983d2b7488412b74b424af6da7078c21a).
- [13] *ANGLE (software)*. 2020. url: [https://en.wikipedia.org/wiki/ANGLE\\_\(software\)](https://en.wikipedia.org/wiki/ANGLE_(software)).
- [14] Tim Berners-Lee et al. *RFC2616: Hypertext Transfer Protocol – HTTP/1.1*. 1999. url: <https://datatracker.ietf.org/doc/html/rfc2616>.
- [15] *Blender (software)*. 2021. url: [https://en.wikipedia.org/wiki/Blender\\_\(software\)#Suzanne](https://en.wikipedia.org/wiki/Blender_(software)#Suzanne).
- [16] Alex Bocharov. *Cloudflare Bot Management: machine learning and more*. 2020. url: <https://blog.cloudflare.com/cloudflare-bot-management-machine-learning-and-more/>.
- [17] Kevin Bock et al. “unCaptcha: a low-resource defeat of recaptcha’s audio challenge”. In: *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*. 2017.

- [18] Dieter Bohn. *Nobody is flying to join Google's FLoC*. 2021. url: <https://www.theverge.com/2021/4/16/22387492/google-floc-ad-tech-privacy-browsers-brave-vivaldi-edge-mozilla-chrome-safari>.
- [19] *Browser Market Share Worldwide*. 2021. url: <https://gs.statcounter.com/browser-market-share>.
- [20] Michele Campobasso and Luca Allodi. "Impersonation-as-a-Service: Characterizing the Emerging Criminal Infrastructure for User Impersonation at Scale". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer I& Communications Security*. CCS '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1665–1680. isbn: 9781450370899. doi: [10.1145/3372297.3417892](https://doi.org/10.1145/3372297.3417892). url: <https://doi.org/10.1145/3372297.3417892>.
- [21] Richard Clayton, Steven J Murdoch, and Robert NM Watson. "Ignoring the great firewall of china". In: *International workshop on privacy enhancing technologies*. Springer. 2006, pp. 20–35.
- [22] The Trade Desk. *UnifiedID2/uid2docs*. url: <https://github.com/UnifiedID2/uid2docs>.
- [23] Mariano Di Martino, Peter Quax, and Wim Lamotte. "Realistically Fingerprinting Social Media Webpages in HTTPS Traffic". In: *Proceedings of the 14th International Conference on Availability, Reliability and Security*. ARES '19. Canterbury, CA, United Kingdom: Association for Computing Machinery, 2019. isbn: 9781450371643. doi: [10.1145/3339252.3341478](https://doi.org/10.1145/3339252.3341478). url: <https://doi.org/10.1145/3339252.3341478>.
- [24] Steven Englehardt and Arthur Edelstein. *Firefox 85 Cracks Down on Supercookies*. 2021. url: <https://blog.mozilla.org/security/2021/01/26/supercookie-protections/>.
- [25] Amin FaizKhademi, Mohammad Zulkernine, and Komminist Weldemariam. "FPGuard: Detection and prevention of browser fingerprinting". In: *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer. 2015, pp. 293–308.
- [26] *Fingerprint Web Server*. url: [https://owasp.org/www-project-web-security-testing-guide/latest/4-Web\\_Application\\_Security\\_Testing/01-Information\\_Gathering/02-Fingerprint\\_Web\\_Server](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/01-Information_Gathering/02-Fingerprint_Web_Server).
- [27] Chris Fredrickson. *Issue 1122019: Entropy exposed by HTMLMediaElement.canPlayType is unmeasured*. 2020. url: <https://bugs.chromium.org/p/chromium/issues/detail?id=1122019>.
- [28] *Gamepad API*. url: [https://developer.mozilla.org/en-US/docs/Web/API/Gamepad\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Gamepad_API).
- [29] *Getting Started with Headless Chrome*, Google Developers. url: <https://developers.google.com/web/updates/2017/04/headless-chrome>.
- [30] *GPU Accelerated Compositing in Chrome - The Chromium Projects*. 2014. url: <https://sites.google.com/a/chromium.org/dev/developers/design-documents/gpu-accelerated-compositing-in-chrome>.
- [31] Kashmir Hill. *'Do not track' privacy tool doesn't do anything*. 2018. url: <https://gizmodo.com/do-not-track-the-privacy-tool-used-by-millions-of-peop-1828868324>.
- [32] Md Imran Hossen and Xiali Hei. "A Low-Cost Attack against the hCaptcha System". In: *arXiv preprint arXiv:2104.04683* (2021).
- [33] *IPv6*. 2021. url: <https://en.wikipedia.org/wiki/IPv6>.

- [34] *IPv6 Statistics*. url: <https://www.google.com/intl/en/ipv6/statistics.html>.
- [35] Michael Kan. *Inside the Pandemic's Biggest Cash Cow: Scalper Bot Networks Hawking Hot Products*. 2020. url: <https://www.pcmag.com/news/inside-the-pandemics-biggest-cash-cow-scalper-bot-networks-hawking-hot>.
- [36] Amin Faiz Khademi, Mohammad Zulkernine, and Komminist Weldemariam. "An empirical evaluation of web-based fingerprinting". In: *Ieee Software* 32.4 (2015), pp. 46–52.
- [37] Anssi Kostiainen et al. *Battery Status API: W3C Editor's Draft 25 September 2019*. url: <https://w3c.github.io/battery/>.
- [38] Pierre Laperdrix et al. *Browser Fingerprinting: A survey*. 2019. arXiv: [1905.01051](https://arxiv.org/abs/1905.01051) [cs.CR].
- [39] Brad Lassey. *Combating Fingerprinting with a Privacy Budget*. 2019. url: <https://github.com/bslassey/privacy-budget>.
- [40] *Media Capabilities*. 2021. url: <https://www.w3.org/TR/media-capabilities/>.
- [41] *Media Capabilities API*. url: [https://developer.mozilla.org/en-US/docs/Web/API/Media\\_Capabilities\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Media_Capabilities_API).
- [42] *Mitigating Browser Fingerprinting in Web Specifications*. 2019. url: <https://www.w3.org/TR/fingerprinting-guidance/>.
- [43] Lou Montulli and David M. Kristol. *HTTP State Management Mechanism*. RFC 2109. Feb. 1997. doi: [10.17487/RFC2109](https://doi.org/10.17487/RFC2109). url: <https://rfc-editor.org/rfc/rfc2109.txt>.
- [44] Keaton Mowery and Hovav Shacham. "Pixel Perfect: Fingerprinting Canvas in HTML5". In: *Proceedings of W2SP 2012*. Ed. by Matt Fredrikson. IEEE Computer Society. May 2012.
- [45] Gabi Nakibly, Gilad Shelef, and Shiran Yudilevich. *Hardware Fingerprinting Using HTML5*. 2015. arXiv: [1503.01408](https://arxiv.org/abs/1503.01408) [cs.CR].
- [46] Dr. Thomas Narten, Richard P. Draves, and Suresh Krishnan. *Privacy Extensions for Stateless Address Autoconfiguration in IPv6*. RFC 4941. Sept. 2007. doi: [10.17487/RFC4941](https://doi.org/10.17487/RFC4941). url: <https://rfc-editor.org/rfc/rfc4941.txt>.
- [47] Nick Nikiforakis et al. "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting". In: *2013 IEEE Symposium on Security and Privacy*. IEEE. 2013, pp. 541–555.
- [48] Nick Nikiforakis et al. "SessionShield: Lightweight Protection against Session Hijacking". In: *Engineering Secure Software and Systems*. Ed. by Úlfar Erlingsson, Roel Wieringa, and Nicola Zannone. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 87–100. isbn: 978-3-642-19125-1.
- [49] M Nottingham, P McManus, and J Reschke. *HTTP alternative services*. url: <https://tools.ietf.org/html/rfc7838>.
- [50] *NPAPI*. 2021. url: <https://en.wikipedia.org/wiki/NPAPI>.
- [51] Lukasz Olejnik, Steven Englehardt, and Arvind Narayanan. "Battery Status Not Included: Assessing Privacy in Web Standards." In: *IWPE@ SP*. 2017, pp. 17–24.
- [52] Lukasz Olejnik et al. *The leaking battery: A privacy analysis of the HTML5 Battery Status API*. Cryptology ePrint Archive, Report 2015/616. <https://eprint.iacr.org/2015/616>. 2015.
- [53] James O'Malley. *Captcha if you can: How you've been training AI for years without realising it*. 2018. url: <https://www.techradar.com/news/>



- captcha - if - you - can - how - youve - been - training - ai - for - years-without-realising-it.
- [54] Performance API - Web APIs: MDN. url: [https://developer.mozilla.org/en-US/docs/Web/API/Performance\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Performance_API).
  - [55] Matthew Prince. *The Trouble with Tor*. 2020. url: <https://blog.cloudflare.com/the-trouble-with-tor/>.
  - [56] *Protecting users on a thriving web (Chrome Dev Summit 2019)*. Chrome Developers, 2019. url: <https://www.youtube.com/watch?v=WnCKlNE52tc>.
  - [57] QUIC - gQUIC. 2021. url: <https://en.wikipedia.org/wiki/QUIC>.
  - [58] reCAPTCHA. url: <https://www.google.com/recaptcha/about/>.
  - [59] Eric Rescorla. *WebRTC IP Address Privacy*. 2011. url: [https://www.w3.org/2011/04/webrtc/wiki/images/d/da/WebRTC\\_IP\\_Address\\_Privacy.pdf](https://www.w3.org/2011/04/webrtc/wiki/images/d/da/WebRTC_IP_Address_Privacy.pdf).
  - [60] *Risk-Based Authentication*. url: <https://www.okta.com/identity-101/risk-based-authentication/>.
  - [61] Iskander Sanchez-Rola, Igor Santos, and Davide Balzarotti. "Clock Around the Clock: Time-Based Device Fingerprinting". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 1502–1514. isbn: 9781450356930. doi: [10.1145/3243734.3243796](https://doi.org/10.1145/3243734.3243796). url: <https://doi.org/10.1145/3243734.3243796>.
  - [62] Allison Schiff. *Google's Message To The Ad Industry: We Won't Build Our Own Third-Party Cookie Alternatives (And We Don't Want You To Either)*. 2021. url: <https://www.adexchanger.com/online-advertising/googles-message-to-the-ad-industry-we-wont-build-our-own-third-party-cookie-alternatives-and-we-dont-want-you-to-either/>.
  - [63] Allison Schiff. *Unified ID 2.0 Is Facing Roadblocks (And Not Just To Do With Google)*. 2021. url: <https://www.adexchanger.com/online-advertising/unified-id-2-0-is-facing-roadblocks-and-not-just-to-do-with-google/>.
  - [64] Justin Schuh. *Building a more private web: A path towards making third party cookies obsolete*. 2020. url: <https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html>.
  - [65] Ory Segal, Aharon Fridman, and Elad Shuster. In: *EU-17*. BlackHat, 2017. url: <https://www.blackhat.com/docs/eu-17/materials/eu-17-Shuster-Passive-Fingerprinting-Of-HTTP2-Clients-wp.pdf>.
  - [66] A. H. Seh et al. "Healthcare Data Breaches: Insights and Implications". In: *Healthcare (Basel)* 8.2 (2020).
  - [67] *Session hijacking*. 2021. url: [https://en.wikipedia.org/wiki/Session\\_hijacking](https://en.wikipedia.org/wiki/Session_hijacking).
  - [68] Suphannee Sivakorn, Jason Polakis, and Angelos D Keromytis. "I'm not a human: Breaking the Google reCAPTCHA". In: *Black Hat* (2016), pp. 1–12.
  - [69] Peter Snyder. *Brave Fingerprinting Protections*. 2020. url: <https://github.com/brave/brave-browser/wiki/Fingerprinting-Protections>.
  - [70] Brave Software. *How to Use Extensions in Incognito Mode*. 2020. url: <https://brave.com/learn/enable-extensions-in-incognito/>.
  - [71] Konstantinos Solomos et al. "Tales of favicons And caches: Persistent tracking in modern browsers". In: *Proceedings 2021 Network and Distributed System Security Symposium* (2021). doi: [10.14722/ndss.2021.24202](https://doi.org/10.14722/ndss.2021.24202).

- [72] L. Stewart et al. *RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication*. 1999. url: <https://datatracker.ietf.org/doc/html/rfc2617#section-3>.
- [73] Erik Sy et al. "A QUIC Look at Web Tracking". In: *Proceedings on Privacy Enhancing Technologies* 2019 (July 2019), pp. 255–266. doi: [10.2478/popets-2019-0046](https://doi.org/10.2478/popets-2019-0046).
- [74] Naoki Takei et al. "Web Browser Fingerprinting Using Only Cascading Style Sheets". In: *2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*. 2015, pp. 57–63. doi: [10.1109/BWCCA.2015.105](https://doi.org/10.1109/BWCCA.2015.105).
- [75] Kazuhisa Tanabe, Ryohei Hosoya, and Takamichi Saito. "Combining Features in Browser Fingerprinting". In: *Advances on Broadband and Wireless Computing, Communication and Applications*. Ed. by Leonard Barolli et al. Cham: Springer International Publishing, 2019, pp. 671–681. isbn: 978-3-030-02613-4.
- [76] Trishita Tiwari and Ari Trachtenberg. "Alternative (ab)uses for HTTP Alternative Services". In: *13th USENIX Workshop on Offensive Technologies (WOOT 19)*. Santa Clara, CA: USENIX Association, Aug. 2019. url: <https://www.usenix.org/conference/woot19/presentation/tiwari>.
- [77] *Unified ID Solution 2.0: The Trade Desk*. url: <https://www.thetradedesk.com/us/about-us/industry-initiatives/unified-id-solution-2-0>.
- [78] Randika Upathilake, Yingkun Li, and Ashraf Matrawy. "A classification of web browser fingerprinting techniques". In: *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE. 2015, pp. 1–5.
- [79] *Usage statistics of web servers*. url: [https://w3techs.com/technologies/overview/web\\_server](https://w3techs.com/technologies/overview/web_server).
- [80] Yves Vanaubel et al. "Network Fingerprinting: TTL-Based Router Signatures". In: *Proceedings of the 2013 Conference on Internet Measurement Conference*. IMC'13. Barcelona, Spain: Association for Computing Machinery, 2013, pp. 369–376. isbn: 9781450319539. doi: [10.1145/2504730.2504761](https://doi.org/10.1145/2504730.2504761). url: <https://doi.org/10.1145/2504730.2504761>.
- [81] Karel Vandendriessche et al. *Imec.Digimeter 2020*. 2021, pp. 37–40.
- [82] *Vibration API*. url: [https://developer.mozilla.org/en-US/docs/Web/API/Vibration\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Vibration_API).
- [83] Luis Von Ahn et al. "recaptcha: Human-based character recognition via web security measures". In: *Science* 321.5895 (2008), pp. 1465–1468.
- [84] *Web technology for developers*. url: [https://developer.mozilla.org/en-US/docs/Web/API/WEBGL\\_debug\\_shaders](https://developer.mozilla.org/en-US/docs/Web/API/WEBGL_debug_shaders).
- [85] *WebGPU*. 2020. url: <https://en.wikipedia.org/wiki/WebGPU>.
- [86] *WebGPU API*. 2021. url: <https://gpuweb.github.io/gpuweb/>.
- [87] *Websockets API*. url: [https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API).
- [88] *WebXR*. url: [https://developer.mozilla.org/en-US/docs/Web/API/WebXR\\_Device\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API).
- [89] Marissa Wood. *Today's Firefox Blocks Third-Party Tracking Cookies and Cryptomining by Default*. 2019. url: <https://blog.mozilla.org/blog/2019/09/03/todays-firefox-blocks-third-party-tracking-cookies-and-cryptomining-by-default/>.

- [90] Shujiang Wu et al. “Rendered Private: Making GLSL Execution Uniform to Prevent WebGL-based Browser Fingerprinting”. In: *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1645–1660. isbn: 978-1-939133-06-9. url: <https://www.usenix.org/conference/usenixsecurity19/presentation/wu>.
- [91] Pengwei Zhan, Liming Wang, and Yi Tang. *Website Fingerprinting on Early QUIC Traffic*. 2021. arXiv: [2101.11871](https://arxiv.org/abs/2101.11871) [cs.CR].