Fachhochschul-Bachelorstudiengang
SOFTWARE ENGINEERING
A-4232 Hagenberg, Austria

Web Browser Fingerprinting

BACHELORARBEIT

zur Erlangung des akademischen Grades
Bachelor of Science in Engineering

Eingereicht von

Janine Denise Mayer

Begutachtet von FH-Prof. DI Dr. Werner C. Kurschl

Hagenberg, Mai 2019

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, May 31, 2019

Janine Denise Mayer

Contents

Abstract

The world wide web is used by billions of people. The majority of users do not realize how easy it can be for websites to gather information about a person to sell it or use it for their own purposes. Though scandals like the Facebook data leak in 2018 reveal how the simple usage of a website can make private data accessable to third parties. One method of information gathering is the so-called web browser fingerprinting. Still unknown to many, this tracking method provides an easy way to spy on users.

A part for solving this problem is to inform internet users about the dangers of tracking online and show ways to prevent the collection of their data. Therefore users need to have a rough idea how their pieces of information get to third parties and need to have a save way to check if they are trackable online.

The objective of this thesis is to provide basic information about web browser fingerprinting. Further it should suggest methods on how to mitigate the danger of being tracked. The development of a prototype should depict how easy it can be to identify and re-identify users online. Further the user should be able to use the prototype to check if they are trackable.

Kurzfassung

Auf der ganzen Welt verstreut gibt es mehrere Milliarden Internetnutzer. Dem Großteil dieser Nutzer ist nicht bewusst, wie einfach es für Webseiten sein kann private Daten zu sammeln und weiterzuverkaufen. Skandale wie jener von Facebook in 2018 zeigen, wie die Nutzung einer Webseite Daten für Dritte zugänglich macht. Eine Art Informationen von Usern zu sammeln ist das sogenannte Web Browser Fingerprinting. Der breiten Masse gegenüber noch sehr unbekannt, stellt diese Tracking Methode jedoch einen einfachen Weg dar Benutzer auszuspionieren.

Ein Teil der Lösung dieses Problems ist es, Internetnutzer auf die Gefahren des Trackings aufmerksam zumachen und ihnen Lösungswege zu bieten, um das Sammeln ihrer Daten zu verhindern. Hierfür müssen User eine ungefähre Ahnung haben, wie ihre Daten zu Dritten gelangen und einen Weg haben um zu prüfen, ob sie online trackbar sind.

Ziel dieser Arbeit ist es die Grundlagen für das Verständnis von Web Browser Fingerprinting zu bereiten. Weiters sollen Wege aufgezeigt werden, um die Wahrscheinlichkeit, eindeutig im Internet identifiziert zu werden, vermindert wird. Mithilfe eines Prototypen soll dargelegt werden, wie simpel es selbst für Amateure sein kann User eindeutig im Netz zu identifizieren. Internetnutzer sollen in der Lage sein mit dem Prototypen zu testen, ob sie einen eindeutigen Fingerprint haben, durch welchen sie getrackt werden können.

Chapter 1

Introduction

1.1 Motivation

In March 2018, the private data from tens of millions of users were leaked by the social media plattform Facebook.(Confessore [27]) This was not the first big leak of user information (e.g. Google Drive in 2014 (Wei [37])), and it certainly will not be the last.

What is not widely known by less technically interested people, is that you do not need to have an account to leak personal information. Nowadays there are various ways of tracking internet users and collection information. From cookies to fingerprinting, there are always new innovations.

By now tracking became part of the everyday life online. Even advertisments use customizations as this strategy promises to lift sales by 10% (Ariker et al. [24]).

There are few methods to check if you are being tracked and even fewer to avoid being so. Trying to avoid malicious websites, blocking third party cookies, using the incognito mode or not using specific services. But afterall none of these choices seems sufficient to grant real privacy. Therefore the best method seems to be, not using the internet at all.

1.2 Objectives

1.2.1 Analysis of different web browser fingerprinting techniques

One of the objectives of this bachelor thesis is to take a closer look as the different web browser fingerprinting techniques, which are currently in use. First, some of the configurations and plugins, which are used most, are introduced to give a better explanation, why the browser leaks information and what it is necessary for. Further, the different methods of fingerprinting are explained and based on them the different techniques. Based on the foundation set in these subchapters the techniques will be shown based on an application scenario.

The most important points which are discussed regarding this objective are:

- used configurations and plug-ins
- fingerprinting methods
- fingerprinting techniques

1.2.2 Prototype

Due to the outcome of the analysis of the different web browser fingerprinting techniques the best technique for the prototype will be chosen. The objective of the prototype is to show how web browser fingerprinting is implemented and how it works. Based on different configurations and plug-ins the prototype will create a hash which will help to re-identify users.

The prototype will be realized in form of a website which will include a fingerprinting script. It's functionality will cover following points:

- reading configurations and plug-ins
- creating a hash
- (re-)identifying users

1.3 Overview

Following chapters discuss the stated content:

Chapter 2 discusses the basics of web browser fingerprinting, amongst others the different methods of fingerprinting. In order to compare them later on to analyse and eventually choose one of the methods to implement the prototype for chapter 4.
Chapter 3 will specify the requirements which are needed to outline a proper prototype.
Chapter 4 explains the implementation and design of the prototype.
Chapter 5 concludes the previous chapter based on test cases and a proper evaluation.
Chapter 6 summarizes the results of this bachelor thesis and states possible prospects.

Chapter 2

Foundation

This chapter will cover the basic knowledge about fingerprinting. First the term fingerprint will be explained and what it can be used for. Further, the threat it poses as well as suggestions on how to reduce a fingerprint will be discussed before covering technologies which can be utilized to track the user. After the base has been layed the chapter closes with the description of the different fingerprinting methods and techniques.

2.1 Fingerprinting

Web browser fingerprinting, also known as device fingerprinting, is the systematic collection of information to identify and later re-identify users. (Doty [5], p.3)(AmIUnique [22]) This data tracking method works by obtaining data from the user's web browser and using this data to create an unique fingerprint "hash". This hash is the key for later re-identifying the user. (Upathilake, Li, and Matrawy [19], p. 1)(Havens [9], p. 4)

With millions of computers and mobile devices in use and even more browsers, this identification method seems really unlikely to work at first sight. Nevertheless, it is an extremely reliable method for correlating data across websites or even web browsers with individual users. (Havens [9], p. 3)

> Example on how it works:
> There are currently over 7.5 billion people populating our planet. Imagine you have to identify a single person. By stating this specific person is male, you already cut the choice in half. Information about the ethnicity and age of this person, as well as the time zone he lives in will narrow down the choice even more.
> It's basically the same with web browser fingerprinting. The gender can be seen as an equivalent of the used operating system, the ethnicity as the browser and age as the browser version. Information about the time zone and used language are easily acquired by the browser and so are more specific details which help narrowing down the choice to one person.

This combination of properties can be obtained using code (time zone, screen resolu-

tion, plug-ins) and also be extracted from HTTP headers (user agent string). (Szymielewicz and Budington [18])

Web browser fingerprinting was first developed short before 2010 and due to its effectiveness quickly gained traction in the tracking industry. (Havens [9], p. 3)
Many of the fingerprinting methods can not be detected by the user, and as it is extremely difficult for users to modify their web browsers in a way that they are less vulnerable to it (AmIUnique [22])(Szymielewicz and Budington [18]), makes it a dangerous tool for deanonymization attacks and maliciously-minded tracking programs. (Havens [9], p.3)

> Example:
> Upon accessing a website, the fingerprinting script determines the user's fingerprint and pushes it to a central data store. This data store is usually a shared data store which is used by multiple websites. So when the user accesses another website, which also has access to this specific shared data store, the user will be recognized and the user profile can be updated. (Havens [9], p. 8)

2.2 Utilisation

In a study conducted in 2013 by Nikiforakis et al. they found that fingerprinting is a part of some of the most popular websites on the internet and that multiple hundred thousands of users are fingerprinted on a daily basis. (Nikiforakis et al. [15], p.6)

The following paragraphs lists different ways in which fingerprinting can be utilized:

Constructive use

- Security authentication
  Using web browser fingerprinting to correctly identify a device which is used to log into an account can be used to re-identify a user and help combat fraud or credential hijacking. (Upathilake, Li, and Matrawy [19], p.4)(Nikiforakis et al. [15], p.2)

  > Example:
  > Services can track the devices used to access an account and inform the user in case an unknown device tries to access it.

Destructive use

- Hacking
  Habits of users can be tracked without their knowledge and attackers can acquire specific knowledge about the user's software setup. (Upathilake, Li, and Matrawy [19], p.4)

> Example:
> With the help of web browser fingerprinting a hacker can acquire knowledge about the users operating system and adjust his hacking method accordingly.

Con- and destructive use

- Identifying criminals
  As a mean of tracking, fingerprinting can be used to track people or even criminals. But as seen in the previous examples this can be used to harm or benefit users.

  > Example: positive use
  > When a user A is harassed or stalked by another user B, the website can use web browser fingerprinting as a mean of blacklisting said user B.

  > Example: negative use
  > Citizens of countries with a strict regime can be blacklisted for expressing criticism against the government.

- Commercial use
  Commercial fingerprinting is used to track people's habits and preferences. The acquired knowledge can be used to either help the user or direct him into a certain direction.

  > Example: positive use
  > A website picks up the user's habit to look for climbing gear and suggests similar products.

  > Example: negative use
  > A website registers that the user recently bought more expensive products and starts to only show items in this certain price range rather than more suitable but cheaper products.

The following paragraphs will summarize the findings of a study conducted by Nikiforakis et al. in cooperation with three commercial fingerprinting companies in order to test multiple websites for the use of fingerprinting scripts.

Nikiforakis et al. used the categorization of TrendMicro and McAfee on a list of 3804 domains. If a domain was neither analysed nor categorized by both used services they were marked as untested and not used. Therefore only 59.2% of the 3804 domains were included in the results of this testing. If one service declared a domain as unsafe and the other stated the opposite, it was accepted as safe. Some categories were given aliases and

gathered in a more generalized category.(Nikiforakis et al. [15], pp.6)

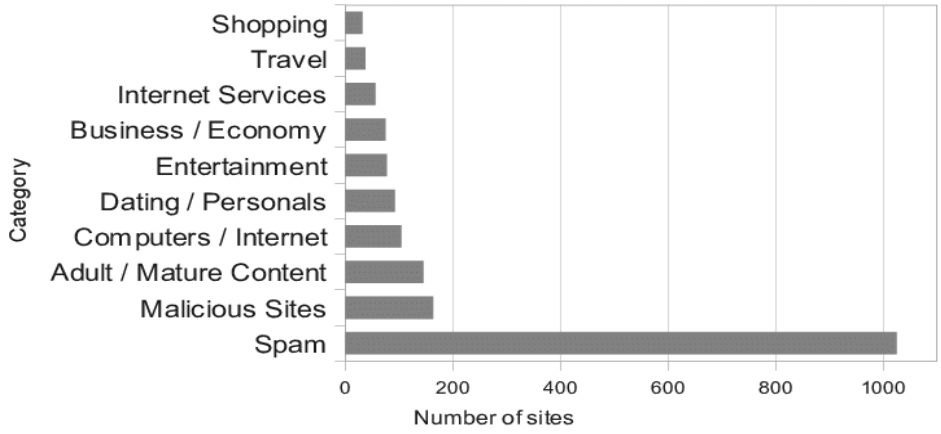

Figure 2.1: Top 10 categories utilizing fingerprinting (Nikiforakis et al. [16])

Figure 2.1 shows 10 categories, 8 of which operate with user subscriptions. These sites usually are interested in preventing hijacking of user accounts and fraudulent activities. The top two categories were identified as malicious (163 domains) or as spam (1063 domains). Nikiforakis et al. noticed that many domains belonging to one of these categories were parked websites which do not include any fingerprinting code. Further, many "quiz" or "survey" websites were discovered which extract a user's personal details. The domains were categorized as malicious if they exploit vulnerable browsers or extract private data from users. (Nikiforakis et al. [15], p.7)

It was discovered that all these sites were found to include, at some point, fingerprinting code provided by one of the three commercial fingerprinting companies.
This observation, paired with the fact that all three studied providers stated that there must be made an appointment in order to acquire fingerprinting services, point to the possibility that fingerprinting companies work with dubious domains in order to expand their fingerprinting database. (Nikiforakis et al. [15], p.7)

2.3 Threat

As seen in the previous section 2.2 web browser fingerprinting can be utilized in destructive ways. This section will take a closer look at the threat fingerprinting poses for the user's privacy and security.

The most tricky part about fingerprinting is, that on the one hand users usually do not know that they are being tracked, and on the other hand they can't do anything to prevent it (see section 2.4). (Upathilake, Li, and Matrawy [19], p.4)

So for example P. Eckersley was able to generate unique fingerprints for 83.6% of 470,161 browsers that belong to users who have a high awareness of privacy concerns. (Eckersley [6], p.2,8)

Even though the fingerprint can change due to modified characteristics, there is a simple algorithm to detect these changes. With heuristics this algorithm was able to re-identify a changed fingerprint with 99.1% accurancy out of 65.56% guesses of changed fingerprints.(Eckersley [6], p.13)

Below are a few threats listed which fingerprinted users might face:

- due to the possibility of being identified some users might face surveillance, risk their personal physical safety or concerns about discrimination due to their internet activities. (Doty [5], p.4)
- the tracking enables collection without clear indications that such a collection is happening without clear or effective user controls. (Doty [5], pp.4)
- Tor enables JavaScript by default and if the user fails to turn it off, even Tor users can be tracked due to the information leaked by JavaScript. (Havens [9], pp.9)

2.3.1 Under General Data Protection Regulation

This threat posed by web browser fingerprinting as depicted above might be reduced by the *General Data Protection Regulation (GDPR)*, which entered into force on May 25th 2018. This regulation imposed by the European Union (EU) intends to cover this kind of hidden data collection, which is used by web browser fingerprinting by forcing companies to prove they have a legitimate reason for the utilization of any means of tracking. (Szymielewicz and Budington [18])

Even though the GDPR avoids specifying technologies, it provides general rules which should keep up with technological development. Due to this regulation browser characteristics are now to be treated like personal data. Personal data has a broad definition, as any information that might be linked to an identifiable individual can be passed as such. Examples for such are not only the IP-Address and MAC-Address of users but also less specific features, including the combination of characteristics which web browser fingerprinting relies upon. (Szymielewicz and Budington [18])

In order to be allowed to use fingerprinting legally the concerned entity has to complete the following steps:

- show that the tracking does not violate the fundamental rights and freedoms of the data subject, including privacy,
- and is in line with reasonable expectations of data subjects
- further, give a legitimate argument for its interest in tracking,
- and share details about the scope, purposes, and legal basis of the data processing with the person subjected to the fingerprinting.

Due to this regulation, the only step the user has to take to avoid fingerprinting is to say "no".

Even though the rules imposed by this regulation seem to help prevent unwanted tracking, it only helps mitigate it. There will certainly be fewer entities making use of fingerprint-

ing, though web browser fingerprinting is not expected to disappear, no matter how high the penalties are on its illegal utilisation.

Anyway, the GDPR only applies on processed personal data of individuals living in the *European Economic Area (EEA)* for commercial purposes, or for any purposes when the behaviour is within the EEA. There will always be companies which either think they can escape the consequences or which claim to have "legitimate interest" in tracking users. Furthermore, no matter how strict regulations are, it always comes down to the user. Due to the plentiful requests for consent as they are found on the web nowadays many users are worn out and do not take a second look at the privacy policy regulated by the GDPR. (Szymielewicz and Budington [18])

2.4 Mitigation

As seen in the previous sections there are many ways of utilizing web browser fingerprinting. Many of these possible applications pose a threat to the user's privacy which brings up the question if it can be avoided. Each of the researched papers about this topic states the same - No. Unfortunately, it is impossible to opt-out fingerprinting. (analytics [2])(Upathilake, Li, and Matrawy [19], p.4)

Even if it is not possible to prevent the possibility of being fingerprinted, there are many suggestions on how to at least mitigate the characteristics which are used to generate an unique fingerprint. Some of these suggestions are stated below:

- Blocking tools which are maintained by regular web-crawls to detect tracking and incorporate blocking mechanisms (Acar et al. [1], pp.11);
  This might work with active fingerprinting methods but not with the passive methods.
- Having Flash provide less system information and report only a standard set of fonts (Nikiforakis et al. [15], p.13);
  This would require Flash developers to take enough interest in this topic to change security settings. It might also cause rendering problems.
- Use private browsing modes or Tor anonymity service;
  Tor is slower and while it disables WebGL it still allows rendering to the <canvas> element which partly exposes the user to canvas fingerprinting. (Mowery and Shacham [13], p.1)
- Browser vendors agree on a single set of API calls to expose to the web applications as well as internal implementation specifics (Nikiforakis et al. [15], p.13);
  This would require all browser vendors to take enough interest in this topic to change their API calls.
- Browser vendors agree on an universal list of "canvas-safe" fonts (Mowery and Shacham [13], p.10)(Boda et al. [3], p.16);
  This would again require the cooperation of all browser vendors. If only a few browsers adapt their settings this might make them more distinct.
- Support WebGL which ignore graphic cards and render scenes in a generic software renderer;

This approach might be good while the performance impact is not. (Mowery and Shacham [13], p.10)

- Require the users approval whenever a script requests pixel data (Mowery and Shacham [13], p.10);
  The reason why the approval is needed might be covered. Further, a casual user will not know the effects of accepting.
- Modification of the user agent string;
  Opinions vary on this ones. While Yen et al. claim that this measure helps, Nikiforakis et al. as well as other authors state otherwise. (Yen et al. [20], p.5) (Nikiforakis et al. [15], p.13) (Eckersley [6], p.4)
- Decrease the verbosity of plug-in versions and the user agent string (Boda et al. [3], p.16);
  This measurement might help to reduce the uniqueness of said characteristics.
- Enable the user to set fake system properties in browser like hiding the operating system, time zone and screen resolution (Boda et al. [3], p.16);
  This measurement might increase the fingerprintability if the real properties leak or are maliciously set.
- Browser extension which automatically blocks active content e.g. NoScript, ScriptBlock (analytics [2])
  Helps against canvas fingerprinting but does not prevent passive methods.
- Use commonly-used web browsers and default settings, including the operating system (analytics [2]);
  Good suggestions as no fingerprinting technique is able to distinguish identically configured devices.

2.5 Critical Technologies

As mentioned in the previous sections scripts which use fingerprinting can read certain data via configuration settings or other observable characteristics from the web browser, which enables them to create a specific fingerprint. (Doty [5], p.3) The following paragraphs will outline some of these configurations.

2.5.1 JavaScript

JavaScript is a web development language for web functionality. (Wood [38]) It allows client-side scripting and gives access to many browser-populated features like the installed plug-ins. (AmIUnique [22]) It is enabled by default in all major browsers and by November 2018 JavaScript was included by 95% of all websites online. (javascript.info [28]) (w3techs [36])

JavaScript can be executed on any device which has a *JavaScript Engine*. (Mulazzani et al. [14], p.2) (javascript.info [28]) This engine is responsible for parsing the script to compile it into machine language which can be exeucted. During this process the engine applies optimizations on every state of the process. (javascript.info [28])

Known JavaScript Engines:

- V8 in Chrome and Opera
- SpiderMonkey in Firefox
- ChakraCore for Microsoft Edge

JavaScript Engines usually implement features which are specified by *ECMAScript* . (javascript.info [28])(Mulazzani et al. [14], p.2)
ECMAScript provides rules, details, and guidlines which have to be regarded for a scripting language in order to be ECMAScript compliant. It is represented by ECMA-262, which is a standard published by Ecma International (responsible for creating standards for technologies). (Aranda [23])

In-Browser JavaScript is seen as "safe" programming language as it only has a low-level access to memory or CPU. Its abilities are therefore limited and it has no direct access to OS system functions. (javascript.info [28])

Nevertheless, can JavaScript also be used to retrieve browser characteristics like the user's time zone and system color. Further there are two properties which can be used to acquire additional information about the user's screen and window.(analytics [2])

The *navigator object* is a possible property for window objects and supported by all major browsers. Informations which can be acquired through the navigator object include: browser name and version, if cookies are enabled, browser language, platform and the useragent (which does not differ from the user agent in subsection 2.5.4)(analytics [2])

The *screen* is a property which holds information about the user's screen. For example the screen width and height, the acutal available display width and height and the color depth which is supported by the user's device.(analytics [2])

When JavaScript is disabled, websites are not able to detect the list of active plug-ins and fonts, and neither will be able to install certain cookies on the targeted browser. The disadvantage of disabling JavaScript is that websites won't always function properly, because it is also used to make websites run smoothly and therefore will impact the browsing experience.(pixelprivacy [33])

2.5.2 Adobe Flash

Adobe Flash is a proprietary browser plug-in which provides different ways of delivering rich media content which is usually not displayed in HTML. (Nikiforakis et al. [15], pp.2)
It used to be the most commonly used video format on the internet but can now be replaced with safer technologies like HTML5 (see subsection 2.5.3). Even though it could be replaced, it is still installed on the vast majority of browsers. (analytics [2])(Nikiforakis et al. [15], p.3)

This plug-in can be used to retrieve sensitive data from the user, for example can it detect HTTP proxies. Besides, has it reimplemented certain APIs which already exist in browers and are accessible through the use of JavaScript (see subsection 2.5.1). (Nikiforakis et al. [15], p.2) This rich programming interface (API) can be used to access system-specific attributes, such as the operating system and its version, a list of fonts, screen resolution and time zone.(AmIUnique [22])(Havens [9], p.5) Furthermore, the API does not provide the same result as the original browser API, but even more detailed return values. (Nikiforakis et al. [15], p.2)

> Example:
> Firefox might return “Linux x86 64” if it is queried about the platform execution while Flash might provide the full kernel version "Linux 3.2.0-26-generic". (Nikiforakis et al. [15], p.2)

Apart from these security risks is Adobe Flash also being criticized for its poor performance, as well as lack of stability. (Nikiforakis et al. [15], p.3) Another vulnerability is its local storage objects which enable so called *zombie cookies*. These zombie cookies populate in various storage locations on the targeted device and re-populate in case one of these storage areas is cleared. These cookies can also be used paired with fingerprinting in order to keep track of changes in fingerprints ("cookie syncing"). (Havens [9], p.8f)

Adobe Flash can be disabled or even uninstalled which might effect the user experience on old websites negatively. (pixelprivacy [33])

2.5.3 HTML 5

HTML5 is the latest version of the Hypertext Markup Language, which is a coding language used to build websites and supported by all major browsers. (pixelprivacy [33])(Marshall [31]) It can be used to build complicated applications, as well as provide animations, movies, music, and the likes. This without the need of any additional software (plug-ins). Summed up, it provides features like geolocation, graphics, video and webapps. (Marshall [31])

Apart from its incredible capabilities there is a slight problem concerning HTML5 video. As it would not be agreed upon which format(s) should be supported, which ended in each browser supporting different HTML5 video formats.
As HTML5 does not include digital rights management technology which is used to prevent the user from copying the content, content owner usually prefer Adobe Flash (see subsection 2.5.2). (Marshall [31])

One of HTML5’s new elements is <canvas> element which is used to draw graphics on a web page.(pixelprivacy [33]) Through this process, it is possible to acquire any differences in the hardware and software configurations due to slight image rendering differences. (AmIUnique [22]) This provides a way to inspect the data with pixel accuracy which later can be used in fingerprinting techniques like described in subsection 2.7.2 (Mowery and

Shacham [13], p.2)

In subsection 2.7.2 *Base64 encoded images* are used which is one of the oldest ways to encode something in html. Base64 encoded images are a part of the HTML and enable the website to display the images without loading. It turns different types of data into a combination of numbers and letters which is safe for HTML. One of the main benefits is that the webpage does not have to load an external ressource which helps the page to load faster. This performance benefit only works if small images are used. (Sexton [35])

2.5.4 User agent

A user agent is a request header field in the hypertext transfer protocol (HTTP) which is used for the communication between browser and website. (Xovi [39]) It is automatically sent with each page call, except if specifically configured not to. (Xovi [39])(Fielding and Reschke [8], p.5.5.3)

The user agent contains the name and version of the used browser. In R. Fielding's work about HTTP, this combination is also called *product identifier*. The product identifiers are listed in the order of their importance, whereas each of them can be followed by one or multiple comments. (Xovi [39])(Fielding and Reschke [8], p.5.5.3) This field value is often called user agent string. (Hoffman [10])

> Example 1:
> Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.102 Safari/537.36 Vivaldi/2.0.1309.37

This string tells the receiving server, that the latest version (2.0.1309.37) of the Vivalid browser is used. In the comments (contained in the brackets) it gives away that the computer runs Windows 10 (Windows NT 10.0) on a 64-bit version (WOW64).

> Example 2:
> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.140 Safari/537.36 Edge/17.17134

Similar to the first example, this browser (Edge with the version number 17.17134) also gives away the operating system and its version (same as before).

As shown in the examples above, not only the used browser name and its version are passed on, but also information about the operating system and sometimes hardware. (Hoffman [10]) (Xovi [39]) The passed-on information is used to display different content in different browsers, on different operating systems or to gather statics (Hoffman [10]) (Arntz [25]) (Xovi [39])

If wanted, changes on the user agent string can easily be made. (Arntz [25]) The problem with this configuration is, that the longer the user-agent field values become, the higher becomes the risk of being identified through for example fingerprinting. Therefore, the user should limit the addition to the user-agent string which can be requested by third parties. (Fielding and Reschke [8], p.5.5.3)

2.6 Web browser fingerprinting methods

2.6.1 Active fingerprinting

Active fingerprinting actively queries information about the client by running JavaScript code or using plug-ins like Adobe Flash which extend the browsers functionality. (analytics [2])((Doty [5], p.6)

Some of the additional characteristics which can be retrieved with this method are:

- user's screen (width, height, resolution)
- window size
- enumerating fonts
- time zone
- plug-ins
- evaluating performance characteristics
- rendering graphical patterns

The only potential disadvantage active fingerprinting provides, is that it can be detected on the client-side in case the user analyses the outgoing data packages, HTML or the JavaScript source code. However, this does not happen in the majority of cases. (Doty [5], p.6)(analytics [2])

Active fingerprinting techniques

- Canvas fingerprinting
- JavaScript Engine fingerprinting
- Cross-Browser fingerprinting
- Browser specific fingerprinting

2.6.2 Passive fingerprinting

In contrast to active fingerprinting methods, passive fingerprinting does not need to execute any code on the client side. This method works based on observable characteristics which are contained in the header data of the IP packet by default. (Doty [5], p.6)(analytics [2]) There is no way for the user to learn if a website is using a passive fingerprinting method and is secretly storing data.

Some of the provided characteristics are:

- cookies

- HTTP request headers
- IP address and port
- character sets (e.g. UTF-8)
- languages

Passive fingerprinting techniques

- Browser specific fingerprinting

2.7 Web browser fingerprinting techniques

As described in section 2.6 there are different methods of fingerprinting. These methods are implemented in different techniques which will be discussed in this chapter.

2.7.1 Browser specific fingerprinting

In 2010, P. Eckersley conducted a study on the project "Panopticlick" which checks how trackable users are. The tracking is done with the help of a fingerprint which is generated from previously collected data. (Upathilake, Li, and Matrawy [19], p.2) This study has shown that 94.2% of the users which use Flash or JavaScript could be distinguished with the help of browser fingerprinting. (Eckersley [6], p.2)

How it works

Browser specific fingerprinting is a technique which only uses browser-dependent features to collect a number of common and less-common known browser characteristics.

This kind of fingerprinting is possible even if the browser only reveals its version and configuration information which is usually enough to create a distinct fingerprint. (Eckersley [6], p.16) Although the retrieved browser information typically includes the installed fonts, plug-ins (including version numbers), user agent, HTTP accept and screen resolution. (Upathilake, Li, and Matrawy [19], p.2)

As this kind of fingerprint is just a concatenation of this information it is easily affected by changes, like upgrades, newly installed features or external system changes. Usually, simple heuristics can be used to adjust the browser specific fingerprint accordingly.(Eckersley [6], p.4) (Upathilake, Li, and Matrawy [19], p.2)

In some cases when the user tries to hide browser features, the opposite happens as the fingerprint is made more distinct. See the example below.

Example:
Even when using a Flash-blocking add-on, the browser still displays Flash in the plug-in list. Due to the add-on the list of system fonts can't be obtained, this creates a distinct fingerprint, although the Flash-blocker was not explicitly detected. (Eckersley [6], p.4)

Like many other web browser fingerprinting techniques, browser specific fingerprinting is not able to distinguish instances of identically configured devices. Further, other than cross-browser fingerprinting, this is a single browser fingerprint technique, which means it can not re-identify a user in another browser as the fingerprint may change across browsers. Upathilake, Li, and Matrawy [19], p.2)

2.7.2 Canvas fingerprinting

Canvas fingerprinting was first mentioned and implemented by Mowery and Shacham in 2012 as part of their work "Pixel Perfect: Fingerprinting Canvas in HTML5".
Any website that runs JavaScript on the user's browser can use the HTML5 <canvas> element and observe its rendering behaviour. There is no need for any special access to system resources. (Mowery and Shacham [13], p.1)(Upathilake, Li, and Matrawy [19], p.2)

The outcome of their research and tests showed that the canvas fingerprint is consistent but can change depending on hardware and software configuration. Only 2 years after its first use, a study conducted by Acar et al. ascertained that canvas fingerprinting is the most common form of fingerprinting with an approximate occurrence of 5.5% in the top 100,000 Alexa websites. (Acar et al. [1], p.5)

How it works

Canvas fingerprinting renders text and WebGL scenes onto an area of the screen using the HTML5 <canvas> element programmatically. Each browser renders differently which helps acquiring a distinctive part for the fingerprint. So for example, out of 300 samples the text font Arial was rendered in 50 distinctive ways.(Mowery and Shacham [13], p.6)

After rendering the text or WebGL scene using HTML5<canvas> the pixel data is read to generate a fingerprint. Through this process a 2D graphic context is obtained and a text or image is drawn to the canvas.(Upathilake, Li, and Matrawy [19], p.2)
The obtained canvas object provides a toDataURL() method which gives a data url consisting of a Base64 encoding of a png image which contains the canvas' content. This Base64 encoded pixel data url is used to create a hash.(Mowery and Shacham [13], p.3)(Upathilake, Li, and Matrawy [19], p.2)

Base64 is an encoding standard which is used when binary data needs to be encoded. Basically it works by splitting the given binary input into groups of 6 bit. Using the 64 alphabet table (see Figure 2.2) each group is depicted by one of the 63 characters in this alphabet. (Josefsson [12])

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

Figure 2.2: Base64 Alphabet (Josefsson [11])

At least the operating system, browser version, graphics card, installed fonts, sub-pixel hinting and anti-aliasing all play a part in the final fingerprint.(Upathilake, Li, and Matrawy [19], p.2)

In their paper K. Mowery and H. Shacham used two types of image comparisons to check if the user's configuration matches a fingerprint: *pixel-level difference* and *difference maps.*(Mowery and Shacham [13], pp.4)

Pixel-level difference works by setting each pixel's color to a specific color. If the pixel's color isn't a transparent pure black it indicates that two pixels don't match and therefore its alpha value is set to 255 to render it completely opaque. (Mowery and Shacham [13], pp.4)

Difference maps work in a similar way, only that this method sets a pixel either black or white, depending on whether they differ. If an image is purely white it indicates that they are identical images. (Mowery and Shacham [13], pp.4)

Canvas fingerprinting can not distinguish identically configured devices and can not fingerprint a user across different browsers.(Mowery and Shacham [13], p.5) Upathilake, Li, and Matrawy [19], p.2)

2.7.3 JavaScript Engine fingerprinting

In 2011 P. Reschl et al. published their first paper about the use of the JavaScript Engine to uniquely identify a web browser and its version. Comparing it to the Panopticlick project (Eckersley [6]) it was stated in the paper that the new approach is multiple times faster and less error prone. (Reschl et al. [17], p.2) Two years later, in 2013, the same team with some additional authors published another paper in which this fingerprint technique was described in more detail.(Mulazzani et al. [14], p.1)

Compared to other methods this technique does not rely on the user agent string (which

can be modified, see subsection 2.5.4) which makes it a more robust fingerprinting technique.(Mulazzani et al. [14], p.1)

How it works

JavaScript Engine fingerprinting works by running test cases of conformance tests like Sputnik and test262 to identify browsers and their major versions. Even though these test suits consist of multiple thousands test cases the browser only needs to fail one particular test which every other browser passed to be identified. Therefore these fingerprinting technique only needs a fraction of a second to be executed. (Mulazzani et al. [14], p.3)

There are two methods which can be used for identification:

For a rather small test set the *minimal fingerprint method* can be used. First test262 is started for each browser in the set and the results are compared. For each browser a test is selected which only this specific browser failed (uniqueness = 1). This browser can be uniquely identified by this test and is therefore removed from the set. This process is repeated for each browser until there is a unique fingerprint for each browser in the set or there is no test case, which only one browser failed. If there is no unique fingerprint for a browser the selection is simply changed.(Mulazzani et al. [14], p.3)

Figure 2.3 shows how the uniqueness of browsers and tests is obtained. The check marks indicate, that the respective browser has passed the test. Each test which has a uniqueness of 1 can be used to uniquely identify the respective browser.

Web Browser	Test 1	Test 2	Test 3	Test 4
Browser 1	✓	×	×	×
Browser 2	×	✓	×	✓
Browser 3	×	✓	✓	×
Uniqueness	**1**	**2**	**1**	**1**

Figure 2.3: Example for the structure of a minimal fingerprint

If the test set is rather large it is more effective to use a *binary decision tree*. This method runs multiple test rounds to determine a web browser and major version. There is no need for a unique fingerprint as the tests split the browsers to identify them. Therefore there is no need to run a test for each browser and version unlike the minimal fingerprint which reduces the total number of executed tests. (Mulazzani et al. [14], p.4)

Figure 2.4 gives a good overview of how such a tree can be structured. The leaf nodes pose as browsers, the inner nodes display the used tests and the edges indicate the success. (Mulazzani et al. [14], p.4)

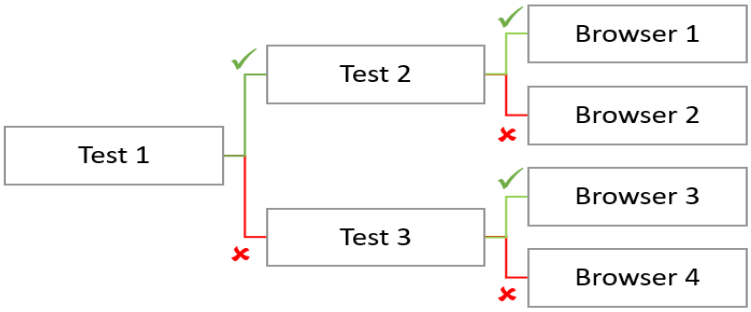

Figure 2.4: Example for the structure of a decision tree

2.7.4 Cross-Browser fingerprinting

In 2011 Boda et al. conducted the first study on cross-browser fingerprinting, utilizing a part of the user's IP address as part of the fingerprint. Even though this technique is not optimal, as the IP address can be modified by e.g. proxies, it remained the only paper published this kind of fingerprinting until Cao et al. published an improved version in 2017. (Boda et al. [3], p.14)(Cao, Li, and Wijmans [4], p.1)

How it works

Cross-Browser fingerprinting uses browser-independent features to track a user regardless of the browser they are using. This can be done by relying only on the use of JavaScript and font detection. (Upathilake, Li, and Matrawy [19], pp.2)

To be able to track a user across different web browsers a set of browser-independent features is used as a basis of identification. Part of this set is for example (Boda et al. [3], p.2)

- networking information (ip address, hostname, TCP port number)
- application layer information (user agent string, name and version of the operating system, extensions)
- information gained by quering (list of fonts, screen resolution, timezone, plug-ins and their version)

The difference to other fingerprints is that the first two octets of the IP address are used, which usually remain constant even if the IP address of the client changes dynamically. The downside is that it may change when switching ISP or other services. (Boda et al. [3], p.5)

Besides, the IP address is modified by proxies for privacy reasons, which changes the unique fingerprint. Boda et al. dealt with that problem saying that the other parameters suffice to identify a user.

Cao et al. improved this method by excluding the IP address, modifying the use of some data and adding some features.(Cao, Li, and Wijmans [4], p.1) This technique forces the

system to perform 36 tasks which return the required information in less than a minute.

This fingerprinting technique is based on different novel operating systems and hardware level features (e.g. from graphic card, CPU, audio stack, and installed writing scripts). Many of these features are exposed to JavaScript over web browser APIs, which are used to extracted them.(Cao, Li, and Wijmans [4], pp.1)

Some improvements Cao et al. made towards Boda et al.'s technique are:

> Improvement 1:
> Change of the screen resolution. In browsers like Firefox and Internet Explorer the resolution changes when the user zooms. Therefore this method takes the zoom level into consideration and normalizes the width and height of the screen resolution. (Cao, Li, and Wijmans [4], p.2)

> Improvement 2:
> The formats DataUrl and JPEG are unstable across different browsers, therefore Cao et al. prefered the use of a lossless format like PNG. (Cao, Li, and Wijmans [4], p.2)

The script sends various rendering tasks (e.g. drawing curves and lines to client side) and also obtains operating system and hardware level information. The client side browser renders and returns the results (images, sound waves) which are converted into hashes. In the meantime the browser collects browser-specific information.(Cao, Li, and Wijmans [4], p.4)

The final fingerprint is generated from a list of hashes which is intertwined with a mash with the help of an "and"-operation. The mask consists of collected browser information, whereas each browser has its own mask. (Cao, Li, and Wijmans [4], p.4)

Contrary to single browser techniques cross-browser fingerprinting has no problem identifying users across browsers. Anyway, it has the same weakness as all fingerprints: identically configured devices. (Upathilake, Li, and Matrawy [19], pp.2)

Chapter 3

Requirements

As part of this bachelor thesis a prototype is composed to show how fingerprinting techniques are implemented. This section first discusses the use case which is covered by the prototype before a distinction between functional and non functional requirements is made and non-goals are identified.

3.1 Use case

A use case is defined to distinguish between needed and obsolete features. The prototype is programmed to fulfill the following use case:

- *User recognition:* The prototype gives the user the chance to check if he has a unique fingerprint. On the users wish the website creates a fingerprint for the user. After the creation the user is informed if he has been recognised or if this was the first visit on the website. The user can try and change his fingerprint, then get back to the prototype and check if the website still recognises him.

3.2 Functional requirements

- *Acquire data for browser fingerprinting:* screen propterties, languages, timezone, plugins, et cetera.
- *Acquire data for canvas fingerprinting:* pixel data of the rendered canvas.
- *Comparision of hashes:* Comparing already existing hashes from the local JSON file with the newly created hash.
- *Save hash:* Saving the created hash with the current date and time to a local JSON file.
- *Feedback for client:* Display a short message if the client has been recognised or not.
- *Compatibility:* The prototype should be able to work on multiple operating systems and in various browsers.

3.3 Non functional requirements

- *Usability:* It is easy for the user to check his fingerprint and the feedback is understandable.
- *Correctness/Reliability:* The program creates the same fingerprint for a user in case nothing has changed.
- *Testability:* It is easy to test the prototype (creating a fingerprint and check back with the given time of the first visit).
- *Information:* The prototype gives a brief overview what the used techniques are about.
- *Protability:* The prototype can easily be distributed to different devices.

3.4 Objectives

- Use at least one fingerprinting technique to create the prototype.
- Test the prototype with multiple web browsers and operating systems.

3.5 Non Objectives

- Use heuristics to check if a fingerprint has been changed.
- Inform the user which part of his hash is unique and which is identical to other users.

Chapter 4

Design and Implementation

This chapter covers the design and implementation of the previously mentioned prototype (requirements see chapter 3). The first section describes the architecture of the protoype with the help of application scenarios. The second section covers programming details and the last section gives a short overview of the visualisation of the prototype.

4.1 Architecture

4.1.1 Used Fingerprinting Techniques

As seen in section 2.7 there are various web browser fingerprinting techniques. To show how easy it can be to program such a technique the prototype is used to implement two of those techniques.
There are various factors which influenced the decision making on which fingerprinting techniques should be used and in the end the decision fell on browser specific and canvas fingerprinting.

Browser specific fingerprinting was chosen as it was the first utilized fingerprinting technique and delivers a good example to show how easy it is to retrieve unique information from a user.

Canvas fingerprinting was chosen as it aquires the information needed for the hash over rendering for example fonts which no other technique does. Further a study conducted in 2016 by S. Englehardt and A. Narayanan detected that 5.1% of the Alexa Top 1000 used canvas fingerprinting scripts . Anyway, this percentage declined the further down the Top Alexa 1 Million they crawled.(Englehardt and Narayanan [7], p.12)

4.1.2 Application scenario

As a mean of explanation a application scenarios will be used to outline the concept this prototype is based on. Further it will give a short overview of how browser specific and canvas fingerprinting work when executing the prototype.

General concept

In general the prototype depicts a simple client-server application.
The main parts of this application scenario are:

- *Servlet*, can be seen as a controller, which manages requests and responses.
- *JavaServerPage* (short JSP), holds HTML for the webpage and JavaScript.
- *web.xml*, is a web application deployment descriptor, which means it defines everything about the application the server needs to know. In this case what the welcoming page is.
- *Ajax*, used for the datatransfer from the website back to the servlet.
- *JSON*, two local JSON files are used to hold the created fingerprints for comparison.

Figure 4.1 is a simple outline of the general sequence of operations.

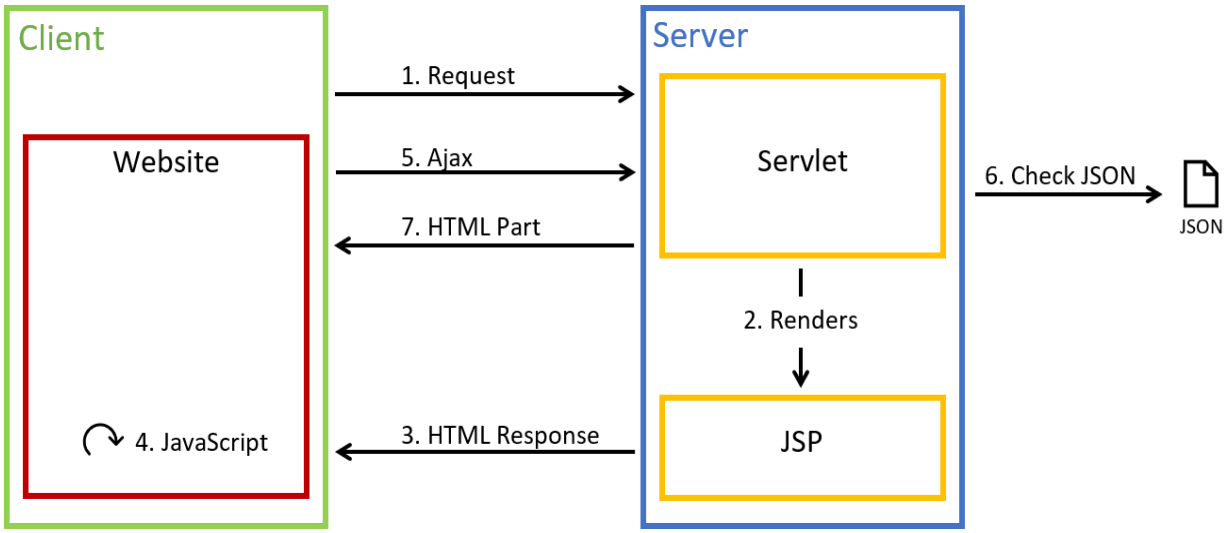

Figure 4.1: Concept of the prototype

1. The client requests the prototype website with the following url:
   http://[ip-address]:8080/FingerprintPrototype/index.jsp
   Due to the servlet the prototype can be accessed from different devices when the server runs, which makes the prototype easily portable.
2. The servlet receives the request and renders the welcoming page which is stated in WEB-INF\web.xml (in this case index.jsp).
3. The JSP is then rendered in the clients browser.
4. The user can click one of the depicted buttons to create his fingerprint. The button triggers a JavaScript method which will lead to create the respective fingerprint. As JavaScript is executed directly in the users browser, JavaScript needs to be enabled in the browser.

5. When the calculation of the fingerprinting hash is done Ajax is used to deliver the new hash to the Servlet.
6. The servlet then compares the newly created hash with already existing ones which are stored in a local JSON file. If the fingerprint is not recognised it is added along with the current date and time to the JSON file.
7. Thereafter the servlet renders a short message in form of static HTML in the JSP on the website of the user. This message informs the user if he has been recognised or else if this was his first visit on the prototype.

Depending on which kind of fingerprint has to be created, the following sections will go into detail with what happenes in step 4 which is described above.

Browser specific fingerprinting

The following Figure 4.2 displays an application scenario which describes what happens when the user choses to create a browser specific fingerprint:

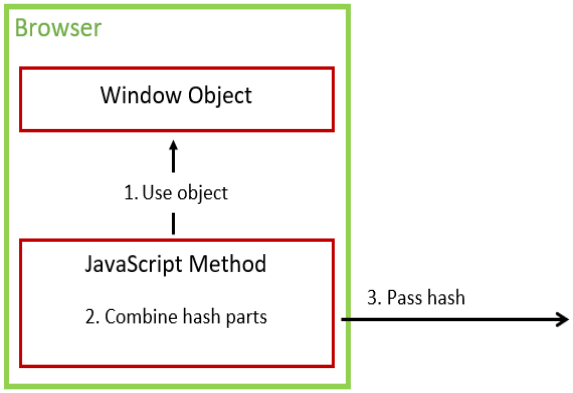

Figure 4.2: Concept of browser specific fingerprinting

1. The browser specific fingerprint JavaScript method uses the window object to retrieve information about the browser (e.g. navigator and screen properties).
2. The acquired information is then combined with further acquired information (e.g. fonts and GPU).
3. The created hash is then passed back to the server (see figure 4.1).

Canvas fingerprinting

The following Figure 4.3 depicts an application scenario which describes what happens when the user choses to create a canvas fingerprint:

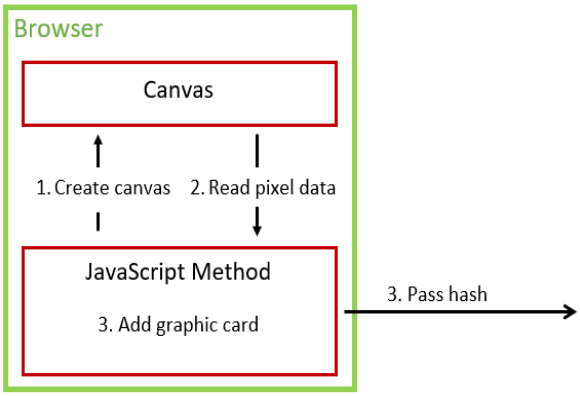

Figure 4.3: Concept of canvas fingerprinting

1. The canvas fingerprint JavaScript method renders a hidden canvas in the browser. The canvas contains different fonts, colors and forms.
2. With the help of the toDataURL() method the content of the canvas is converted into pixel data which is retrieved by the JavaScript method.
3. The retrieved pixel data is then concated with GPU details (only in this prototype).
4. The created hash is then passed back to the server (see figure 4.1).

4.2 Implementation details

As shown with the help of the application scenario (see figure 4.1) the prototype is a simple client server application.

Following subsections cover the implementation details about the prototype. How exaclty the fingerprints are composed will be outlined in subsection subsection 4.2.5 and subsection 4.2.6.

4.2.1 Servlet

The servlet *(Fingerprinting.java)* is working like a controller which is taking care of requests and responses. The main focus lays on the doPost() method which is in charge of the hash comparison. The newly created hash is compared with all existing hashes from a local JSON file. Here is to considered that two different files are In case that the local JSON file does not contain the new hash, a new JSON object with the current date, time and the hash is added to the file.
Here is to be considered that two different files are used to save the hashes. This is due to the fact that the two fingerprinting techniques create completely different hashes which can not be compared to one another.

At the end of the method a feedback for the user is created which then is rendered in the JSP. The following code snippet shows how a PrintWriter is used to send HTML code as a response to the browser.

```
PrintWriter out = response.getWriter();
if(existingUser) {
  out.println("<p style=\"color:red;\">");
  out.print("I remember you!");
  out.println("</p>");
  out.print("You first visited this prototype on:" + firstVisit);
}
else {
  out.print("This is your first visit!");
}
```

4.2.2 JSP

The JavaServerPage *(index.jsp)* contains HTML as well as JavaScript code. The HTML is used to render the prototype in the users browser while JavaScript is executed upon a button click which starts the fingerprinting process (see subsection 4.2.5 and subsection 4.2.6). Apart from the bare creation of the fingerprints, JavaScript is also using Ajax to forward the generated hash back to the servlet.

```
$.ajax({
  type : 'POST',
  data : {
    method : 'canvas',
    hash : hash
  },
  url : 'Fingerprinting',
  success : function(result) {
    $('#result1').html(result);
  }
});
```

As seen in the code snippet above, Ajax posts two different data items. "method" is used to make a distinction between the transfer of a canvas (like above) or a browser specific fingerprinting hash. The "hash" data simply contains the created hash variable.

On success of the transfer the response of the servlet will be rendered in the HTML component with the id "result1". As seen below the response will be rendered where the button, which called the fingerprinting method, was previously rendered.

```
<div class="panel-footer" id="result1">
  <button type="button" class="btn btn-info"
  onclick="fingerprint_canvas();">Check Canvas Fingerprint</button>
</div>
```

4.2.3 Fonts method

Javascript's sandboxed inside the browser and doesn't have privileges to read from the clients disk for security reasons.

You need to have your own list of fonts to check, then you have an array of installed fonts by checking each of the list to see which one is installed.

The difference in widths will tell you the availability of the fonts installed on the client's computers because the browser will fall back to its default font. So you probably need to do some invisible testing for text widths to determine if a font is installed.

https://stackoverflow.com/questions/3597682/how-to-iterate-the-installed-fonts-using-javascript

As installed fonts in a browser can not be directly queried, a work around is used to check which fonts are installed. This workaround simply renders specifically stated fonts and checks through the rendering if they are installed.

To execute this workaround a detector is needed. The detector used in this prototype was implemented by *Latil Patel*(Patel [32]).

As already mentioned, the detector can only check explicitly queried fonts wherefore the following method is needed (which uses Patels detector):

```
1  function get_fonts() {
2    var fonts = [ "Arial", "  Helvetica", "Verdana", "Comic Sans",
3    "Windings", "Webdings", "Georgia", "Rotterdalle", "Sweet Cake",
4    "Coldiac", "Kilauea", "Blacker", "Rubik", "Cormorant" ];
5    var fontDetector = new Detector();
6    var fontExist = "";
7    for (var i = 0; i < fonts.length; i++) {
8      fontExist += fontDetector.detect(fonts[i]) + " - ";
9    }
10   return fontExist;
11 }
```

The detector simply renders all fonts which are called by the method above and checks if they can be rendered. If so, they are available and a respective boolean is returned and added to a string which is eventually the return value of the *get_fonts()* method.

4.2.4 GPU method

To retrieve the users grapic processing unit a function implemented by *Lesha Rubinshtein* (Rubinshtein [34]) is used. As seen in the code snippet below the function creates a canvas (like the canvas fingerprinting technique) and uses it to retrieve GPU specific information.

```
1  function get_gpu() {
2    var canvas = document.createElement('canvas');
3    var gl;
4    var debugInfo;
5    var vendor;
6    var renderer;
7
8    try {
9      gl = canvas.getContext('webgl')
10     || canvas.getContext('experimental-webgl');
11   } catch (e) {
12   }
13
14   if (gl) {
```

```
15        debugInfo = gl.getExtension('WEBGL_debug_renderer_info');
16        vendor = gl.getParameter(debugInfo.UNMASKED_VENDOR_WEBGL);
17        renderer = gl.getParameter(debugInfo.UNMASKED_RENDERER_WEBGL);
18      }
19
20      return renderer;
21    }
22
```

4.2.5 Browser specific fingerprinting

As outlined in subsection 2.7.1 the browser specific fingerprint consists of multiple pieces of information which are retrieved from the user. The method used in this prototype retrieves its pieces of information mainly from a browser object called *window*.

Window

The window object is a property which is supported by all main browsers. It is automatically created by the browser and holds information about the window (javatpoint [29]). It is therefore an essential information source for the browser specific fingerprinting technique.

The following properties are queried in the prototype:

- *screen:* allows to query information like the screen heigth and width or color depth.
- *navigator:* enables the retrieval of information of different configurations and plugins (see below).

```
1   if (window.screen) {
2     var screenAvailWidth = window.screen.availWidth;
3     var screenAvailHeight = window.screen.availHeight;
4     var screenWidth = window.screen.width;
5     var screenHeight = window.screen.height;
6     var screenColorDepth = window.screen.colorDepth;
7   }
```

As documented in the code above, it is a simple property call to retrieve individual measurements from users.

Navigator is one of the properties of the window object and the main source of the information needed for creating the browser specific fingerprint in this thesis (javatpoint [29]). It is read-only and can be accessed via window.navigator or simply navigator(al. [21]).

Information queried with the help of window.navigator (al. [21]):

- *language:* returns the preferred language which is used in the browser UI (null if unknown).
- *languages:* returns all languages which are known by the browser.
- *user agent string:* returns the user agent string of the browser (see subsection 2.5.4).

- *cookieEnabled:* returns a boolean if a cookie can be set or will be ignored.
- *doNotTrack:* returns 'yes' or 'no' depending on the users doNotTrack setting.
- *hardwareConcurrency:* returns the number of available logical processor cores.
- *maxTouchPoints:* returns the value of maximal concurrent touch contact points of the used device.
- *plugins:* returns an array which holds all installed plugins in the browser.

All of the queries properties return a boolean, integer or string except for plugins. Plugins returns a pluginarray which is not a JavaScript array but still possesses a length property (al. [21]).

```
1  var pluginCount = navigator.plugins.length;
2  var plugins = "";
3
4  for (var i = 0; i < pluginCount; i++) {
5    var version = navigator.plugins[i].version ? navigator.plugins[i].version
6    : "x";
7    plugins += navigator.plugins[i].name + ", "
8    + navigator.plugins[i].description + "(" + version
9    + "), " + navigator.plugins[i].filename + " | ";
10 }
```

As the code above shows how easily specific information about the plugins can be retrieves to be used for the creation of a browser specific fingerprint.

Additional to the information queried from the window properties, the prototype also includes the users gpu (graphic processing unit) (subsection 4.2.4) and font check (subsection 4.2.3) in the fingerprint.

After retrieving the necessary information mentioned above, the prototype concats the pieces of information and the created string serves as fingerprinting hash.

Fingerprint

As mentioned above the hash which is created by the prototype consists of a concatenation of all retrieved information about the user.

The example below was retrieved from a device running Windows 10 as an operating system. The fingerprint tells that a Vivaldi Browser was used, the window has the measurements 1280 x 720 and the used lanugage is German. The booleans concatinated with " - " are generated by the font method and depict which fonts are installed.

```
"firstVisit":"18.03.2019 (04:09)",
"hash":"Mozilla\/5.0 (Windows NT 10.0; WOW64) AppleWebKit\/537.36 (KHTML, like Gecko)
    Chrome\/72.0.3626.122 Safari\/537.36 Vivaldi\/2.3.1440.60 | 1218 | 720 | 1280 |
    720 | 1280 | 24 | false | de-DE | de-DE,de,en-US,en | -120 | -60 | true | null |
    4 | 0 | 2 | Chromium PDF Plugin, Portable Document Format(x), internal-pdf-viewer |
    Chromium PDF Viewer, (x), mhjfbmdgcfjbbpaeojofohoefgiehjai | true - true - true -
    false - false - true - true - false - false - false - false - false - false -
    ANGLE (Intel(R) HD Graphics 520 Direct3D11 vs_5_0 ps_5_0)"},
```


Figure 4.4: Example for a browser specific fingerprint

4.2.6 Canvas fingerprinting

As outlined in subsection 2.7.2 the canvas fingerprint consists of pixel data which is retrieved from a canvas in the users browser. This canvas is rendered through the use of JavaScript in the users browser and not allocated to HTML thus appearing hidden. Therefore the user does not see the rendered fonts, colors and forms.

```
1   var canvas = null;
2   var canvasInput = null;
3   var hash = null;
4   var allSigns = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ`~1!2@3#4$5
    %6^7&8*9(0)-_=+[{]}|;:',<.>/?";
5
6   try {
7     // create canvas
8     canvas = document.createElement('canvas');
9
10    // fill canvas
11    canvasInput = canvas.getContext('2d');
12    canvasInput.textBaseline = "top";
13    canvasInput.font = "14px 'Arial'";
14    canvasInput.textBaseline = "alphabetic";
15    canvasInput.fillStyle = "#f60";
16    canvasInput.fillRect(125, 1, 62, 20);
17    canvasInput.fillText(allSigns, 2, 15);
18  }
```

The code above was written based on the two websites *browserleaks* (browserleaks [26]) and *darkwavetech* (jkula [30]).

As seen a canvas is created and an input variable (canvasInput) is prepared. The '2d' written in the getContext method leads to the creation of a CanvasRenderingContext2D obext which represents a two-dimensional rendering context. This depicted in the code snipped above, this context is used to render not only fonts but also colors and rectangles. To render text various different letters and signs are used to recover pixeldata which is as diverse as possible.

After filling the canvas with input *canvas.toDataURL()* is called which returns the Base64 encoded representation of pixel data. This string is then concatenated with the retrieved GPU which creates the used canvas fingerprint.

Fingerprint

As mentioned above the canvas fingerprint mainly consist of Base64 encoded pixel data. Ad Base64 encoded data is not readably for the human eye (see Figure 4.5), the retrieved GPU (subsection 4.2.4) is concatenated at the beginning of the pixel data.


```
"firstVisit": "18.03.2019 (04:09)",
"hash": "ANGLE (Intel(R) HD Graphics 520 Direct3D11 vs_5_0 ps_5_0) |
        data:image\/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAACWCAYAAABkW7XSAAAgAElEQVR4XuxdB3gUZfr\/fbMlvVISIEh
        HVGoiSQDBAKGIvaBiOxUp+j\/bnQiiSLCD3lnuTimi6NkLVlQgIBYgCSShqSBVaQmBkJ5sdme+\/\/N+M9\/u7LCJgQsKd7vP4yPZ+e
        Yrb\/m9dWYZgp8gBYIUCFLgNKEAO032GdxmMlKATwRvxulOm6nYfATl\/bThVuCNBhl4mjPwRLYfBKwToVrwnlOBAkHAOhW48DvvIQh
        YvzPBg8slGwWCgNVspDx9JgoC1unDq+BO\/SnQdMCaNLcPON4FwzWYN3njf0TI2xYkQNFWgbN7sGDi0t+cS44HeoDzLlAXfjX+fVNlw
        Pt+a58T590FzkeJOcJrunrPRJMldL7fmvM3D3BqDWgIsJbyc\/AChuI9LEAEc51amz6B3RTzaGSwv6IzLxFnilzg8sn7hPmjwPhXx0w
        bSL7M8idv4Gy0V3ZvfD0CoTXvgbExx87XxHFC\/oyxDelNn\/xuzJ\/0gnedCXNfAmOTTeseBuOZXh2lvTNtMRR+xzF6a9YFmkA\/wl
        K\/+eXE+tizsGDy7WhoTqEnLBuc5wn9Cqsd773HShgfTV8KuJ71uvH36QFYxEBodzUKVJIgJwoujdl3onOegJL9Hrf8rwDWCxiGpfxs
        7GKt8Bx\/D6MX\/OAPWFaZksAD\/CoUkz4SPMwAJRUTeFQom\/c+5QU\/A+xVYHaD+L6hcVamN0XevUafr\/LuNdB+fy\/AstKE9mIG
        uYYAi\/MyQMnyo5u\/AdABOiBg+RZtaczvQ2tDaTt3LtjbsUNh5k9bh+DggW5z\/YhFGwSeP+ZeuXl5jawYY527d8l7KmP4wpGM46N5
        C+alDXivac7oqBJcOOa5r50htbY9e\/qm5uReGepyRWwFx6cA72n2miz79HllhlW5+da7vnXXhZ\/74UcP9hk48IMn27b5ceyqb245p
        3v33KwuXXJHbNo0Kj8394pnhEVj\/BvyvoYNe3V2hzMKL16xcmK3X385529gytMAWgqvT9XuhN22BOQFAj666eulA4gxWWCLlfSz9v4
        WUreg+8D5QON+\/+v+PNsKrs0G2FgvLTRMFfxgbBw4nwtuy+qhHCjaikTx9WT+DV5ib4G8q9GM2Aec12kF\/tTuI6zPuwTMFSKu02cj
        knANn4B32QLM5UPEd\/Ia\/ZsAYpPWDgdZDEaxH3EXVooxEjik50aez624CX\/FclzDJmIGlviNfQnnY6b2OWYpF2EV\/xsSWIXfPP\
        /k72CMcifkGaQujOGbvd5hNQ\/BlZiAu\/AlPo7qAq3\/RqDrL\/PUcExfeA5KBRBBu+tPN\/z1rdCI6mvlHGVlCZ3c7tDIlil\/+Sp
        n3dhnN23I\/FhGArevQ2eV4VEGRFdXx7QqO9rmnMjIo\/e889rT75B30iph16uXXzonkzG0k\/M1dZxZn7dvS9uwctUtyda9mcds\/m
        Fo\/Jrvr9kk9W9iPmYzjrNpDJ3BVRceltB29y3zXlywjTysvn2WPZ6W\/uFYKHhlXjK+FXOZdMEOtdtb7zxxhsPpWnnNlQ93hIJt81P
        wrHdNA3wmTp78s8sV3vmlV59NlnNu35H27cqVt84B5x\/QfiYVYAhXcVdxcefE7BW3\/Xz9DdM3A0icn2LIIklqAFDbxBlL+qctviAh
        cefL3vVIpjU8AbDYpHZbs8dc9GzXmrrorW+89vRon8Vp0BoAPiBg2R07FO7IzFzwa21dzMI333jibkhrpFui56ApGXh5QrFhmfS\/Fe
        0aALf7rulubPcueTcTYG3cMKouN\/fyRK8HFWguQ7gUmzry7fceS6+ri\/qXQF6rGyrCPJbdtWve6qFDF9YcPtTx3Y8+eeAp0i0x\/j
        dCQgKlhFY7b\/z2+xuT9x8483zhRhtgLQFr+YrJyft+PfsHsd8QV6QIbwm4pCtOIEMfstIScP3d\/Td8YwXdGvnbcPkD3a\/aioyl9b
        MlQAtYPIHn2XtjCEykUktwkSHhg53mYneSB9\/lXdkgYBXxGNzDrvYCihkgtrEE\/MQTBZjJ78nLkeBD63yMPl6glPMUsWgvICbyChH
        OkWc0iv3gnYcAiP42fwhIM3Ev3uCveK\/Rd3fw67AYc\/F+73Asi0lCy26bFjvt9W\/P648PzIAVEl49WADZU6+7EFqzoH2nH166YMQ
        \/rj9c2j5+8YcP9SK5nTRpwmAw\/Akcr4n7AUjFrPc41yl65YUhXsACsuUYUswzOm7Mzjh\/URHT+KLX3vjbdQHHmQ9kAlPv3ghk5cf
        kNdl+2+RKAlEw7DMDwqRluIr2W16W8PE77z5y68kCrOSUJTcuy544YN+vZ39pBisoWD1\/3jyVQkICuYYAq3PHwhfO6ZV9b8v4fQVOZ
        90Hgm66zkSFh5WdP2LEvPDExF1Hq+ui8\/wByOQwyYjy8oQedbWRrVq33rNm5+5zP1258pappLTkgTCGkprqmDSyMrFxB998443Z7WX
        8OzEf9zKOYTSlpto8y7MnJO75pd\/bpFR0TXU7RxcXdR4YHlH2TURkqWvt6mvO2bp9kMiNyXtLSjr0sinusvhWB+6a99L8sNS0D5\/t
```


Figure 4.5: Example for a canvas fingerprint

The enormous amount of input in the canvas resulted in an extraordinarily long output string. The longer the fingerprint the more data is available for comparison and to discover differences. As the fingerprint is multiple pages long it has been croped for the depiction above.

4.3 Visualisation

Even tough the focus of the prototype lay on the creation and comparison of fingerprints, the prototype should be easy to use for the user. For the layout a few CSS files were used and each fingerprint is plainly explained (see Figure 4.6).

The usage is simple as only a button needs to be pressed and the user receives feedback immediately. The feedback depends on the creation and comparison (see Figure 4.7).

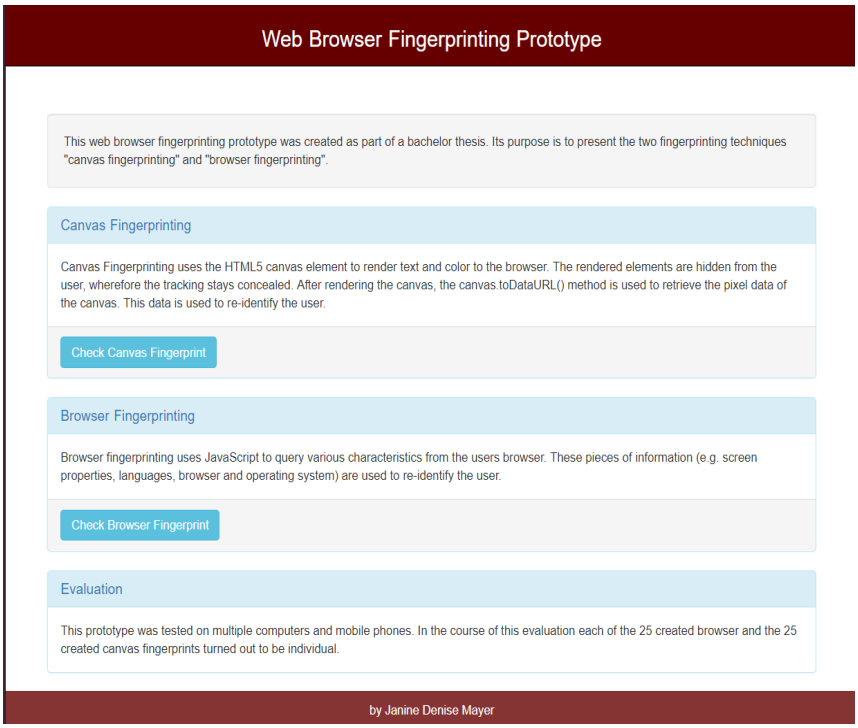

Figure 4.6: Prototype design, fingerprints can be created

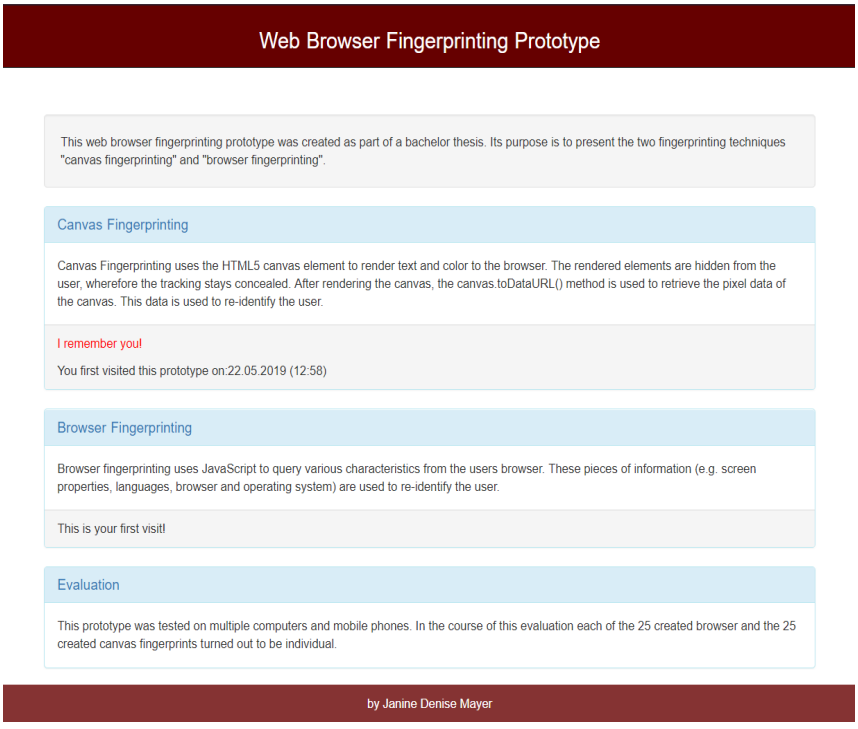

Figure 4.7: Prototype displays feedback for the user

Chapter 5

Testing and Evaluation

This chapter depicts how the implemented fingerprinting prototype was tested and what outcome was achieved.

5.1 Test cases

Two testcases were tested on each fingerprinting technique to assure that the creation and comparison of the fingerprints work properly.

Those test cases are:

- creation and comparison of a new fingerprint
- creation and comparison of a known fingerprint

The test procedure of both test cases is the same:

- start server for prototype
- open prototype in browser
- make sure JavaScript is enabled
- click on button of wanted technique
- check response text below button

The key difference is what knowledge prototype already possesses of the user. When testing with a new fingerprint, the prototype must recognise that the hash is not part of its fingerprint collection and save the newly created fingerprint. Further the correct response has to be displayed.

When testing with a known fingerprint, the prototype must recognise the fingerprint when comparing it to the collection. Therefore it may not add it again and still display the correct user response.

These two test cases have been tested with each of the devices stated in section 5.2.

5.2 Tested devices

As stated in the requirements (chapter 3), the test cases mentioned in section 5.1 have been tested on multiple browsers and operating systems. Both fingerprinting techniques (browser specific fingerprinting and canvas fingerprinting) have been tested on each device mentioned in the following subsections.

5.2.1 Computers

The test cases have been executed on 8 computers with multiple browsers. Each of the computers mentioned in Table 5.1 is a 64 bit system type and has executed the prototype in browsers mentioned in Table 5.2.

Nr.	Operating System	Version	Graphics Processing Unit
1	Windows 10 Home	1803	NVIDIA GTX 1050
2	Windows 10 Home	1803	Intel(R) HD Graphics 520
3	Windows 10 Enterprise	1803	NVIDIA GeForce GTX 960
4	Windows 10 Enterprise	1803	Intel(R) UHD Graphics 630
5	Windows 10 Pro	1803	NVIDIA GeForce MX 130
6	Arch Linux	5.0.2	Mesa DRI Intel(R) HD Graphics 630 (Kaby Lake GT2)
7	macOS Majave	10.14.3	Intel Iris Pro OpenGL Engine

Table 5.1: Tested computers

Browser	Version	Tested devices
Microsoft Edge	18.17763	1
Microsoft Edge	17.17134	2, 3, 4, 5
Google Chrome	73.0.3683.86	1, 4, 7
Google Chrome	73.0.3683.75-2	6
Google Chrome	72.0.3626.121	2, 5
Google Chrome	63.0.3239.84	3
Mozilla Firefox	66.0.2	1, 7
Mozilla Firefox	65.0.2	2, 4
Mozilla Firefox	60.5.2esr	3
Safari	12.0.3	7

Table 5.2: Tested browsers with computers

5.2.2 Mobile Phones

Due to the easy distribution of the prototype the test could also have been executed on multiple mobile phones (see Table 5.3) and mobile browsers (see Table 5.4).

Nr.	Operating System	Graphics Processing Unit
1	Android 9.0	Adreno (TM) 630
2	Android 9.0	Mali-G72
3	Android 7.0	Adreno (TM) 506
4	Android 7.0	Mali-T830
5	iOS 12.1.4	Apple A12 GPU

Table 5.3: Tested mobile phones

Browser	Version	Tested devices
Google Chrome	74.0.3729.157	4
Google Chrome	73.0.3683.90	2, 3
Safari	12.1.4	5
DuckDuckGo	7.16.1	5
DuckDuckGo	5.26.0	4
DuckDuckGo	5.19.0	1

Table 5.4: Tested browsers with mobile phones

5.3 Results

- *Quantity:* Due to the execution of the test cases with the devices and browsers over 25 fingerprints could be acquired per technique.
- *Correctness/Uniqueness:* Each of the acquired fingerprints was unique and could be used to re-identify the user.
- *Compatibility:* The protoype can be executed successfully on each device and in each browser mentioned in section 5.2.
- *Portability:*The test cases can be executed on multiple devices, no matter if desktop computer, laptop or mobile phone.
- *Weakness:*
  - *Zoom level:* Both fingerprints change in case the user created the fingerprint with a different zoom level, as the prototype does not take the zoom level into consideration.
  - *Browser update:* In case of a browser version update the browser specific fingerprint changes, though the canvas fingerprint does not necessarily change.
  - *Cross browser:* As non of the techniques works across browsers the user can not be re-identified in case he visisted the prototype before in another browser.
  - *Plugins:* In case the user installs or de-installs a browser plugin the browser specific technique can not re-identify the user. In case the plug-in does not influence the graphic this does not concern the canvas fingerprint.

Taken everything into consideration the prototype has a quote of 100% in recognising known users with the same configurations. This has been tested with 13 different devices

in 5 different browsers (with 16 versions, see section 5.2).

In case the users configuration have been changed since the fingerprint has been created, the user cannot be re-identified, as configuration alterations have not been taken into consideration for this prototype.

Chapter 6

Summary

This chapter summarizes the results of this bachelor thesis and gives an outlook on possible additions which can be added to the prototype.

6.1 Result

An objective of this thesis was to outline existing fingerprinting techniques including used configurations and techniques. In section 2.5 used technologies have been introduced before a distinction between active and passive fingerprinting methods was made (section 2.6). Following this, the different fingerprinting techniques have been outlined and explained in section 2.7. This objective was concluded with two application scenarios which depicted how browser specific fingerprinting and canvas fingerprinting work (see item 4.1.2 and item 4.1.2).

The other objective of this thesis was to design and implement a prototype which involves the implementation of at least one fingerprinting technique. After setting out specific requirements for the development (see chapter 3) the analysis which techniques should be implemented was made. Based on the chosen techniques the prototype was designed and implemented (see chapter 4). Conclusive to the implementation, the finished prototype has been thoroughly tested with multiple devices (see chapter 5).

6.2 Prospect

As the sole purpose of the prototype has been to depict the implementation of at least one fingerprinting technique truly is only a prototype. There are a number of improvements which can be done to convert this prototype into a proper fingerprinting application:

- *Zoom level:* The zoom level of a browser is a characteristic which can be taken into consideration, as therefore the program will be able to re-identify a user when the zoom level differs from the original zoom level.
- *Heuristics:* Heuristics can be used to check if the users fingerprint has been changed since the first visit. This can help to recognize simple browser updates and new or

updated browser plugins.

- *Information:* The program could give the user a feedback which characteristics have been similar to other users and which are the characteristics which help differentiate between other users. Further tips on how to change those characteristics could be displayed.

References

Literature

[1] Gunes Acar et al. "The web never forgets: Persistent tracking mechanisms in the wild". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2014. URL: https://securehomes.esat.kuleuven.be/~gacar/persistent/the_web_never_forgets.pdf (cit. on pp. 8, 15).

[2] Web analytics. *Browser fingerprints: the basics and protection options*. Tech. rep. 2017. URL: https://www.1and1.ca/digitalguide/online-marketing/web-analytics/browser-fingerprints-tracking-without-cookies/ (visited on 12/08/2018) (cit. on pp. 8–10, 13).

[3] K. Boda et al. "User tracking on the web via cross-browser fingerprinting". In: *Nordic Conference on Secure IT Systems*. Springer. 2011. URL: https://pet-portal.eu/files/articles/2011/fingerprinting/cross-browser_fingerprinting.pdf (cit. on pp. 8, 9, 18).

[4] Yinzhi Cao, Song Li, and Erik Wijmans. "(Cross-)Browser Fingerprinting via OS and Hardware Level Features". In: *Proceedings of Network & Distributed System Security Symposium (NDSS)*. 2017. URL: http://yinzhicao.org/TrackingFree/crossbrowsertracking_NDSS17.pdf (cit. on pp. 18, 19).

[5] N. Doty. *Mitigating Browser Fingerprinting in Web Specifications*. Tech. rep. May 2018. URL: https://w3c.github.io/fingerprinting-guidance/ (cit. on pp. 3, 7, 9, 13).

[6] Peter Eckersley. "How unique is your web browser?" In: *International Symposium on Privacy Enhancing Technologies Symposium*. Springer. 2010 (cit. on pp. 6, 7, 9, 14–16).

[7] S. Englehardt and A. Narayanan. "Online tracking: A 1-million-site measurement and analysis". In: ACM. 2016. URL: http://randomwalker.info/publications/OpenWPM_1_million_site_tracking_measurement.pdf (cit. on p. 22).

[8] R. Fielding and J. Reschke. *Hypertext transfer protocol (HTTP/1.1): Semantics and content*. Tech. rep. 2014. URL: https://tools.ietf.org/html/rfc7231#section-5.5.3 (cit. on pp. 12, 13).

[9] R. Havens. "Browser Fingerprinting: Attacks and Applications". 2016. URL: http://www.cs.tufts.edu/comp/116/archive/fall2016/rhavens.pdf (cit. on pp. 3, 4, 7, 11).

[10] C. Hoffman. *What is a Browser's User Agent?* Tech. rep. 2016. URL: https://www.howtogeek.com/114937/htg-explains-whats-a-browser-user-agent/ (cit. on p. 12).

[11] S. Josefsson. *The base16, base32, and base64 data encodings*. Figure 1 in Josefsson [12]. 2006 (cit. on p. 16).

[12] Simon Josefsson. *The base16, base32, and base64 data encodings*. Tech. rep. 2006. URL: https://www.rfc-editor.org/rfc/pdfrfc/rfc4648.txt.pdf (cit. on pp. 15, 40).

[13] K. Mowery and H. Shacham. "Pixel perfect: Fingerprinting canvas in HTML5". *Proceedings of W2SP* (2012). URL: https://hovav.net/ucsd/dist/canvas.pdf (cit. on pp. 8, 9, 11, 15, 16).

[14] M. Mulazzani et al. "Fast and reliable browser identification with javascript engine fingerprinting". In: *Web 2.0 Workshop on Security and Privacy (W2SP)*. Citeseer. 2013. URL: http://www.ieee-security.org/TC/W2SP/2013/papers/s2p1.pdf (cit. on pp. 9, 10, 16, 17).

[15] N. Nikiforakis et al. "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting". In: *Security and privacy (SP), 2013 IEEE symposium on*. IEEE. 2013. URL: https://seclab.cs.ucsb.edu/media/uploads/papers/sp2013_cookieless.pdf (visited on 10/27/2018) (cit. on pp. 4, 6, 8–11, 40).

[16] N. Nikiforakis et al. *Cookieless monster: Exploring the ecosystem of web-based device fingerprinting*. Figure 3 in Nikiforakis et al. [15]. 2013 (cit. on p. 6).

[17] P. Reschl et al. "Efficient browser identification with JavaScript engine fingerprinting". In: *Proc. of Annual Computer Security Applications Conference (ACSAC)*. 2011. URL: https://publik.tuwien.ac.at/files/PubDat_202737.pdf (cit. on p. 16).

[18] K. Szymielewicz and B. Budington. *The GDPR and Browser Fingerprinting: How It Changes the Game for the Sneakiest Web Trackers*. Tech. rep. EFF, 2018. URL: https://www.eff.org/de/deeplinks/2018/06/gdpr-and-browser-fingerprinting-how-it-changes-game-sneakiest-web-trackers (cit. on pp. 4, 7, 8).

[19] Randika Upathilake, Yingkun Li, and Ashraf Matrawy. "A classification of web browser fingerprinting techniques". In: *New Technologies, Mobility and Security (NTMS), 2015 7th International Conference on*. IEEE. 2015. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7266460 (cit. on pp. 3, 4, 6, 8, 14–16, 18, 19).

[20] T. Yen et al. "Browser fingerprinting from coarse traffic summaries: Techniques and implications". In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2009. URL: https://www.cs.unc.edu/~reiter/papers/2009/DIMVA.pdf (cit. on p. 9).

Online sources

[21] jpmedle et al. *Navigator*. URL: https:/developer.mozilla.org/en-US/docs/Web/API/Navigator (visited on 04/11/2019) (cit. on pp. 28, 29).

[22] AmIUnique. URL: https://amiunique.org/faq (visited on 10/22/2018) (cit. on pp. 3, 4, 9, 11).

[23] Michael Aranda. *What's the difference between JavaScript and ECMAScript?* URL: https://medium.freecodecamp.org/whats-the-difference-between-javascript-and-ecmascript-cba48c73a2b5 (visited on 12/13/2018) (cit. on p. 10).

[24] M. Ariker et al. *How Marketers Can Personalize at Scale*. URL: https://hbr.org/2015/11/how-marketers-can-personalize-at-scale (visited on 11/23/2015) (cit. on p. 1).

[25] P. Arntz. *Explained: user agent*. 2017. URL: https://blog.malwarebytes.com/security-world/technology/2017/08/explained-user-agent/ (cit. on pp. 12, 13).

[26] browserleaks. *Device Fingerprinting*. URL: https://browserleaks.com/canvas#how-does-it-work (visited on 02/23/2019) (cit. on p. 30).

[27] Nicholas Confessore. *Cambridge Analytica and Facebook: The Scandal and the Fallout So Far*. URL: https://www.nytimes.com/2018/04/04/us/politics/cambridge-analytica-scandal-fallout.html (visited on 04/04/2018) (cit. on p. 1).

[28] javascript.info. *An introductin ot JavaScript*. URL: https://javascript.info/intro (visited on 12/13/2018) (cit. on pp. 9, 10).

[29] javatpoint. *Javatpoint*. URL: https://www.javatpoint.com/ (visited on 05/14/2019) (cit. on p. 28).

[30] jkula. *Device Fingerprinting*. URL: https://www.darkwavetech.com/index.php/device-fingerprint-blog?disp=posts&paged=8 (visited on 08/31/2017) (cit. on p. 30).

[31] Gary Marshall. *HTML5: what is it?* URL: https://www.techradar.com/news/internet/web/html5-what-is-it-1047393 (visited on 12/13/2018) (cit. on p. 11).

[32] Lalit Patel. *Font Detector*. URL: https://www.lalit.org/wordpress/wp-content/uploads/2008/05/fontdetect.js?ver=0.3 (visited on 03/15/2019) (cit. on p. 27).

[33] pixelprivacy. *Browser Fingerprinting*. URL: https://pixelprivacy.com/resources/browser-fingerprinting/ (visited on 10/15/2018) (cit. on pp. 10, 11).

[34] Lesha Rubinshtein. *Webgl detect gpu*. URL: https://gist.github.com/oboshto/abd1414811a30bda3b02dc4112c44a71 (visited on 03/15/2019) (cit. on p. 27).

[35] Patrick Sexton. *Base64 encoding images*. URL: https://varvy.com/pagespeed/base64-images.html (visited on 12/13/2018) (cit. on p. 12).

[36] w3techs. *Usage of JavaScript for websites*. URL: https://w3techs.com/technologies/details/cp-javascript/all/all (visited on 12/10/2018) (cit. on p. 9).

[37] Wang Wei. *Google Drive Vulnerability Leaks Users' Private Data*. URL: https://thehackernews.com/2014/07/google-drive-vulnerability-leaks-users_9.html (visited on 07/10/2014) (cit. on p. 1).

[38] Adam Wood. *JavaScript*. URL: https://html.com/javascript/ (visited on 12/13/2018) (cit. on p. 9).

[39] Xovi. *User Agent*. URL: https://www.xovi.de/wiki/User_Agent (visited on 10/15/2018) (cit. on p. 12).

List of Figures

List of Tables