INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Masterthesis

On detecting Web-Tracking

Thomas Müller

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Masterthesis

On detecting Web-Tracking

Thomas Müller

Supervising professor: Prof. Dr. Dieter Kranzlmüller
Supervisors: Dr. Michael Schiffers
Dr. Nils gentschen Felde

Date of Submission: 13. July 2015

I assure the single handed composition of this master's thesis only supported by declared resources.
Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Munich, 13. July 2015

. . . . . .
(Signature of the Thesis Candidate)

Web-Tracking is an incredible big business focusing on the collection of as many user data as possible and use it - among other scenarios - for hand-crafted advertising. Trackers are constantly seeking to improve their tracking mechanisms in order to be able to gather more user-based data. The most recent developments of this research are fingerprinting techniques: Browser- and Canvas-Fingerprinting. Fingerprinting techniques differ to the de facto standard for implementing Web-Tracking - Cookies - in several aspects. Firstly, it is very hard to notice that a web site is fingerprinting the user. Secondly, even if the user knows that he is being fingerprinted, it is very hard to effectively block it. Lastly, fingerprinting techniques enable the trackers to recognize users, which consequently leads to even more user-based data, because this enables them to track users among different web sites. Using the collected user data facilitates the trackers to create detailed user profiles, which clearly threatens the privacy of users. In order to increase privacy being endangered by Web-Tracking, the first step required is to be able to detect the use of fingerprinting techniques. This is the basic task this thesis sets out to solve. The detection part is realized using proven detection mechanisms from different fields like Data Mining or Knowledge Discovery in Databases. The results of the thesis show that Canvas-Fingerprinting is reliably detectable with each of those classifiers. The more general Browser-Fingerprinting is way harder to detect, but the best classifiers still managed to achieve a very high success rate. Consequently, it is safe to say that the developed system is capable of detecting the use of fingerprinting techniques on a web site and can therefore serve as a first step towards a system which can be used to increase privacy by mitigating fingerprinting techniques.

Contents

List of Figures

List of Tables

1. Introduction

Nowadays, the Internet is omnipresent and life without it seems to be impossible for a lot of people. At the latest with the wide distribution of Smartphones, it can be considered a fundamental part of the daily life. Modern communication is inconceivable without the Internet and even the classic SMS is replaced by modern messengers such as WhatsApp [1]. The Internet, however, is used for so much more: people use it for shopping, they search for things they are interested in, they read the news or just use it for plain entertainment. Considering all this convenient use cases, one aspect goes unnoticed: the more the Internet is used, the more personal information is revealed.

The perception changed in 2013 when Edward Snowden published confidential material concerning programs (PRISM, Tempora [2, 3]) of several intelligence agencies - such as the NSA - which aim at observing and collecting personal communication. The publication of these documents had an enormous impact because of the magnitude and impact programs, which actually put the whole population under general suspicion. Ever since the first publication, more and more details became available and the population has started to realize that everything they are using the Internet for is actually observed and stored by the governments. These happenings have led many people to develop a new conscious concerning security and privacy.

While the news that governmental agencies like NSA, BND or GHCQ are tracking the user's behavior on a large scale led to a lot of turmoil, most people seem to simply not know that they have been tracked for a very long time by several companies. Tracking the user and collecting as much information about him as possible is the basis of a lot of web businesses. One way to acquire personal data is usually Web-Tracking. It describes the process of observing the user's behavior when surfing on web sites. This technique also enables the tracker to trace the user among different web sites which results in trackers being able to collect a vast amount of personal data, for example consumer behavior, political interests, financial status or health.

The information gathered by Web-Tracking is highly valuable, especially for the advertisements industry. Using the information enables companies in this industry to handcraft advertisements. The better the advertisements are, the higher the chances that users click on them, which directly results in higher revenue. In order to maximize their gain, companies are constantly trying to improve their tracking mechanisms in order to make them harder to block and to improve their capabilities to recognize single users which directly results in more personal data, which can be used yet again to improve clicking rates. Nevertheless, targeted advertising is only one purpose of user-based data. Its designed use is multi layered. For instance, Web-Tracking can also be used to create extremely detailed user-profiles containing a lot of sensitive data.

The various opportunities for use are the foundation for quite a few companies, which have built their core business around user-based data. This industry is constantly developing advanced and better methods, so called tracking mechanisms, to collect even more personal data. One category of these advanced tracking mechanisms consist of fingerprinting techniques, in particular Browser- and Canvas-Fingerprinting. Those two techniques enable trackers to recognize users on different web sites and extract highly sensitive information. At the same time, the techniques are often performed unnoticed by the user and are hard to block, which adds to the risky nature of these techniques from data security perspective.

This thesis wants to shed some light on Web-Tracking and add to greater transparency by focusing on the detection of the usage of such fingerprinting techniques. The idea is to utilize traditional tracking mechanisms from the fields of Data Mining and Knowledge Discovery in Databases and apply them in the area of Web-Tracking. The thesis aims to dynamically detect, if a web site tries to track a user or not. For this purpose, a system has been developed and prototypically implemented. The remainder of this thesis is structured as follows.

Chapter two explains the fundamentals concerning Web-Tracking and two of its advanced techniques, namely Browser- and Canvas-Fingerprinting. Additionally, it provides the basic operating principles of three relevant detection mechanisms: Naïve Bayes, Decision Tree and ExtraTrees.

Chapter three will focus on related work and will reveal the current state of research in the fields of Browser- and Canvas-Fingerprinting implementation as well as detection, thereby exposing the research gap. In particular, it will be shown that the idea to use traditional tracking mechanisms in order to be able to detect the usage of fingerprinting techniques has not been done before and is completely new idea. This Chapter will also involve possible counter-measures to fingerprinting techniques.

In Chapter four, the basic concept of the thesis will be presented. By showing the basic, wished for flowchart of the planned system, relevant components actuators and communication paths are defined. Those are used to derive a fundamental architectural design. The design is then discussed in detail with every component being illustrated profoundly. Moreover, this Chapter tries to question the architecture with having security and privacy in mind. Furthermore, two different variants to deploy the system will be presented here.

Chapter five will concentrate on the implementation and evaluation. At first, one Section for each of the proposed components will both describe the according part in detail and outline different ways to implement. The following Sections will then describe, how the evaluation of the developed system will be performed dependent on the classifier. Different test scenarios will be illustrated and measurements of each tested classifier will be presented.

Chapter six discusses the measurements. First, it will be shown that the training data collected for this thesis provides differences in terms of used features. After summarizing the measurements, it will be explained, how the results of the tests come into being, followed by a Section that presents an alternative way of testing. With the help of this alternative, it will be shown that the measurements are sound. Lastly, possible improvements to enhance the detection rate of the classifiers will be depicted. Chapter seven concludes.

2. Fundamentals

This Chapter explains basic fundamentals of the thesis. As the title already suggests, it exists of two parts: Web-Tracking and detection. The Web-Tracking part will be covered in Sections 2.1 and 2.2. Those Sections will give an overview of Web-Tracking in general and shed some light on why it is such an important topic. Section 2.3 will illustrate ways to implement Web-Tracking, in particular fingerprinting mechanisms. With Browser- and Canvas-Fingerprinting two techniques from this field will be demonstrated in detail. The second important part of the thesis concerns detection. Therefore, Section 2.4 will concentrate on detection mechanisms used in areas like Data Mining or Knowledge Discovery in Databases. Three different methods will be presented: Naïve Bayesian classifiers, Decision Trees and Extra Trees, where each method will be elucidated profoundly. Moreover, their basic way of functioning will be explained by using some simple examples.

2.1. Web-Tracking in General

Web-Tracking describes the process of tracking user behavior in the Internet by third parties, so called *trackers*. Usually, users do not realize that they are being tracked when surfing the Internet. Mostly, trackers are not the providers of the web site being visited by the user. Of course, web site providers may also track their users. In fact they might even need to implement tracking mechanisms in order to ensure fundamental features such as a shopping basket.

While there are legitimate reasons for web site providers to track their users on their web site, the question arises why third parties, the trackers, are so interested in the user's behavior. Their main goal is the collection of as much user-specific data as possible. This data is incredibly precious to companies, because it represents the user's interests and these interests in turn are the key to targeted advertising, which greatly improves the chances that users click on advertisements and consequently generate money for companies.

Such business procedures go along with problems, in particular the fact that the users have never agreed to such an information gathering. They don't have a clue, what purposes the data serves and they never approved the processing of this data.

Figure 2.1 shows the basic functionality of Web-Tracking. When visiting a web site, the user receives everything relevant for displaying this web site correctly, for example images, text, JavaScript files and stylesheets. Yet, there can also be parts in the response, that belong to the trackers and are connected to Web-Tracking. Those parts may contain pictures, tracking-pixels or similar things that need to be requested from the tracker's server. This request requires the establishment of a connection to the tracker and also contains a *referer-header*[1] that involves the currently visited web site. By reading this header field, the tracker is able to determine, which web site the user has been visiting. If the tracker has the ability to recognize a user, he is also able to create profiles for this very user.

Although one may assume that recognizing users can be a difficult task, there also exists a pretty trivial method to do so. There are more web site providers than one might think, which utilize URL parameters for sensitive, personal data, for example the name. The results are URLs like `http://www.illness.com/cancer.html?name=thomas?surname=mueller`[2]. If there are requests to trackers sent from this URL, the referer header contains it and the tracker automatically does know about the name and the surname of the user. As demonstrated in [5], about 48% of the examined web site providers use identifiers in the URLs, which can be exploited by trackers.

[1] The *referer-header* is an HTTP header that contains the address of the web page, this request stems from [4].
[2] Please note, that this is a fictional URL. Nevertheless, similar URLs are out there.

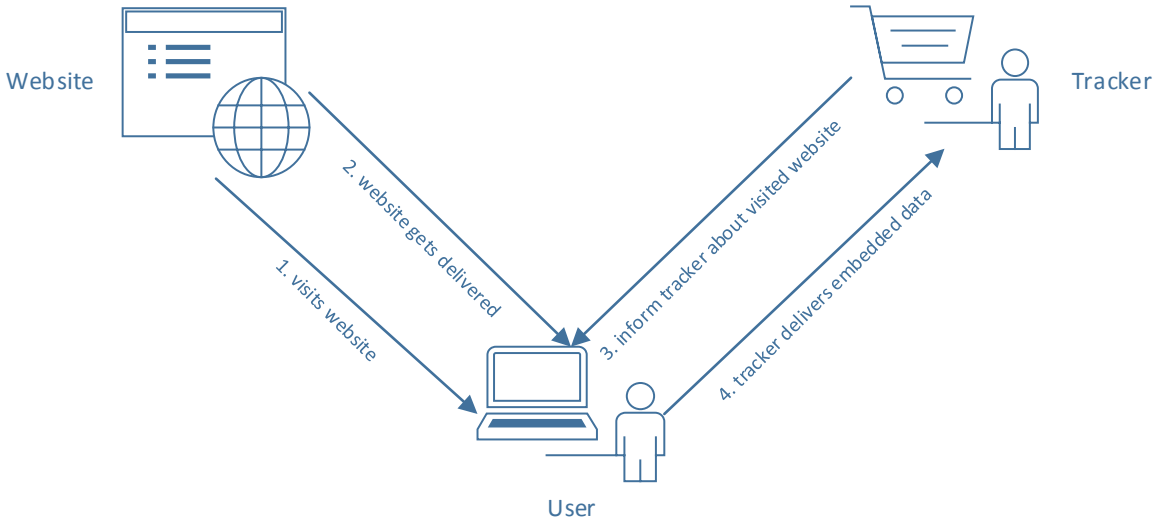

Figure 2.1.: Functionality of basic Web-Tracking.

Names of the users, however, are not the only interesting information, but only a piece that might be of use to the trackers next to other characteristics such as - among others - age, gender, race, zip code, income, marital status, education, income, drug abuse, consume behavior and health concerns [6].

2.2. Cross-Domain-Tracking

In order to tap the full potential of Web-Tracking, trackers have to spread their tracking software not only to one domain, but to as many different domains as possible. This enables the trackers to get user data from different domains, and therefore data from different kinds of sources. For example, a tracker may be able to track the articles read by a user on a web page of a news magazine and he can comprehend his consumer behavior. If a tracker is in the position to track a user across different web sites, it is called *Cross-Domain-Tracking*. Combining the information gained from different domains can greatly improve the quality of the user profile. Generally speaking, the more information trackers can get for an user, the more precise the profile becomes.

Implementing *Cross-Domain-Tracking* requires the trackers to be able to recognize users and to deploy software that is capable of doing so on every domain they wish to track users on.

Figure 2.2 shows the basic way of functioning of *Cross-Domain-Tracking*.

For each visited web site, the user does unknowingly establishes a connection to the tracker's server and consequently informs him about the visited web site. The tracker can recognize the user and extend the respective profile.

It is pretty obvious that trackers try to deploy their software on web sites, which have a huge user base. This enables the trackers to reach as many users as possible. Therefore, it is not surprising, that on some web pages, more than 50 different trackers have implemented their tracking mechanisms [7].

The collection and processing of user-based data is the foundation of an enormous, worldwide market, whose volume might very well exceed several billion dollars. While the exact volume is not known, the dimension is pretty realistic, considering published numbers, which only cover parts of this market. According to [8], only the Internet advertising revenue in United States equaled 49.5 billion dollars in 2014.

The main reason for the market's rapid development has been the increasing number of sources for user-based data. Nowadays, almost everybody possesses a Smartphone, televisions get "smarter" and with the recent launch of the Apple Watch, technology devices are everywhere, at any time, providing information for as well

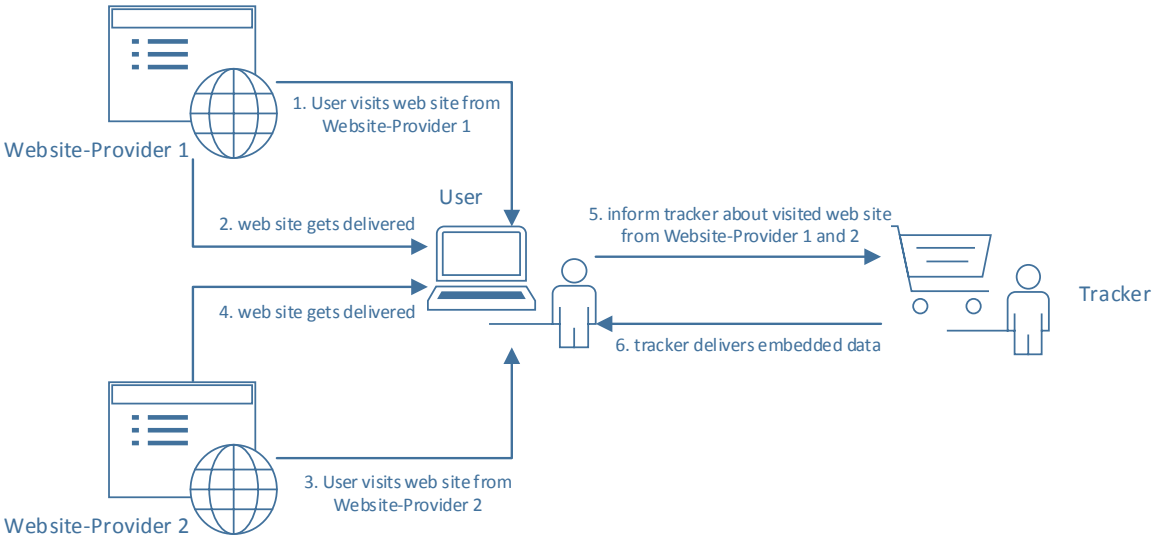

Figure 2.2.: Functionality of Cross-Domain-Tracking.

as gathering data from us. Right now, the trend is to own many devices that are able to communicate with each other via the Internet. The author of [9] shows in great detail, what information will be collected in the future.

From an economic perspective, the last paragraphs gave some first indications about the large, already existing Web-Tracking market and its enormous potential. Having in mind the current development in mobile computing, one might dare to say, that it has almost no upper limit.

Looking at the technical components that made such an development possible, the outstanding technique for tracking people always has been cookies. Cookies are basically small text files, which are stored on the client-side computer. They were developed by Lou Montulli in 1994 [10] and ever since have evolved to an absolutely fundamental principle in modern web applications. Originally, cookies were implemented in order to rule out one of the greatest weaknesses of the HTTP protocol: statelessness. Before cookies were created, web servers were not able to maintain a state for a client. This means, that features such as a basket shop in an online shop, which seem totally normal today, could not be realized in the past. With cookies, this becomes possible: whenever a cookie is set on the client, it is then sent to the server along with every following request. This enables a web site to implement a state.

Cookies are omnipresent in the Internet, but trackers are researching new technologies to implement tracking. Two of the most interesting ones which have a huge potential are called *Browser-Fingerprinting* and *Canvas-Fingerprinting*. In this thesis, those two will be observed in-depth. The following Sections set out to describe them in detail.

2.3. Fingerprinting Techniques

As mentioned before, cookies have been the standard technique used for tracking users. However, cookies are quite easy to block: one can just adjust the browser to drop all cookies and not accept new ones. As recent studies showed, there are other technique in the wild that make blocking a lot trickier: *fingerprinting techniques*. Those are another possibility to implement Web-Tracking and two of them - Canvas- and Browser-Fingerprinting will be demonstrated in the following Sections.

2.3.1. Browser-Fingerprinting

Browser-Fingerprinting is a fingerprinting technique which can be used to implement Web-Tracking. The first part of this Section will give a general overview about this technique. This part will refer to [11], which is one of the first publications with a particular focus on this topic and also present Panopticlick - the project that has been executed as part of the herein before mentioned publication. In order to give a deeper insight, this Section will illustrate the *fingerprintjs* library, one of the few open-source ones which implements Browser-Fingerprinting. The library will serve as an example and helps to explain the functioning of this tracking mechanism.

Panopticlick or: How Unique Is Your Web Browser?

Browser-Fingerprinting is a technique that exploits the browser's functionality to provide rich information about the browser and its underlying system. Browsers are nowadays complex and powerful pieces of software, which are used all the time. Moreover, they supply several different settings and properties, for example the screen width and height or the color depth. Browser-Fingerprinting utilizes the fact that these settings and properties are almost always slightly different on each computer. Therefore, the question arises whether the differences are big enough to be indicative for distinguishing computers.

Peter Eckersley was the first to examine the usefulness of Browser-Fingerprinting on a large scale [11]. He examined the effectiveness of Browser-Fingerprinting algorithms by developing an algorithm that was capable of fingerprinting a browser and deploying it on the web site `http://panopticklick.eff.org`. He spread the link to this software among colleagues and friends which ended in 470.161 different users[3] visiting the web page and therefore participating in the experiment.

But what settings and properties are suitable to be used as part of a fingerprint? Table 2.1 shows, what variables Peter Eckersley used in his fingerprinting algorithm. As one can see, he only used eight different values to fingerprint a browser. Those values were either already sent alongside the HTTP request (User Agent, HTTP ACCEPT headers, cookies enabled?) or are transferred via AJAX[4] (screen resolution, timezone, browser plugins, plugin versions and MIME types, system fonts, partial supercookie test) to the server. The fingerprint itself is than created by concatenating each value. Note that each part of a fingerprint will be called *feature* throughout this work.
Those features provide more information than one might assume at first. For example, it's easy to find out if JavaScript is disabled on the client side. If this is the case, the return values for `video`, `plugins`, `fonts` and `supercookies` are default ones. Another example concerns Flash. If the client uses a Flash blocking add-on, the `plugins` list will show Flash, but it won't be possible to detect the system's fonts using Flash. There are a lot more features that might be suitable to be used as part of a fingerprint. Peter Eckersley did choose to not take those additional features, because he was either not aware of this feature, did not have the time to implement it, did not think that this feature is resilient enough (for instance the IP addresses) or the feature can not be collected without the user's agreement.

Consisting of only eight different features, one might assume that the uniqueness of the fingerprint might not be given in most of the cases. Contrary to that expectation, the results of Peter Eckersley's study showed that about 83.6% of all browsers observed had an unique fingerprint. Moreover, 94.4% of all browsers with either Adobe Flash or a Java Virtual Machine enabled, are unique. Those numbers are impressive and show that Browser-Fingerprinting is capable of recognizing computers by creating surprisingly precise fingerprints.

When observing the features presented in Table 2.1, one might realize that most of them are not persistent, meaning they might change when time passes. For example the list of installed plugins is not fixed, but can change when the user decides to install a new add-on or to delete an existing one. This leads to the question, if such changes influence the uniqueness of the fingerprint. There are a lot of diverse scenarios which can lead to changing features - for example an upgrade of the browser, installing a new font or using an external monitor, which leads to a different screen resolution. While it is true that changing features results in different fingerprints, Peter Eckersley shows the possibility to estimate if a fingerprint is an altered version of

[3] The visitors of the web site were informed about the Browser-Fingerprinting.
[4] This means that these values are selected using JavaScript on the client and then sent to the server via AJAX (**A**synchronous **J**avaScript **a**nd **X**ML).

Table 2.1.: Features used in the fingerprinting algorithm from Peter Eckersley.

Variable	Source	Remarks
User Agent	Transmitted by HTTP, logged by server	Contains Browser micro-version, OS version, language, toolbars and some- times other info.
HTTP ACCEPT headers	Transmitted by HTTP, logged by server	
Cookies enabled?	Inferred in HTTP, logged by server	
Screen resolution	JavaScript AJAX post	
Timezone	JavaScript AJAX post	
Browser plugins, plugin versions and MIME types	JavaScript AJAX post	Sorted before collection. Microsoft Inter- net Explorer offers no way to enumerate plugins; we used the PluginDetect JavaScript library to check for 8 common plugins on that platform, plus extra code to estimate the Adobe Acrobat Reader version.
System fonts	Flash applet or Java applet, collected by JavaScript/AJAX	Not sorted.
Partial supercookie test	JavaScript AJAX post	We did not implement tests for Flash LSO cookies, Silverlight cookies, HTML5 databases, or DOM globalStorage.

a fingerprint seen previously in [11]. For this purpose, he developed a pretty simple algorithm which is based on heuristics. Despite its simplicity, the algorithm's guesses were correct in 99.1% of the cases - although the algorithm was in no way optimized, as Peter Eckersley stated.

In Figure 2.3 one can see the fingerprint for standard hardware[5]. Surprisingly, the relatively standard hardware used is unique among 5.344.051 million browser fingerprints tested so far, which is, considering the small amount of features in this algorithm, highly impressing.

In order to be able to determine the value of a feature, Eckersley [11] introduced the term *entropy*. The higher the entropy of a feature, the more suitable it is as part of a fingerprint, because it provides more information. This can be easily seen in Table 2.2. The possible values for the feature *cookies_enabled* are not that diverse, which results in a low entropy of 0.353. The installed fonts and their ordering however provide a lot of information.

As Table 2.2 shows, the installed add-ons (15.4 bits) and fonts (13.9 bits) have the highest entropy values, meaning that these features are considered the strongest ones among the features set. Because of that, they are explained in greater detail in Sections 2.3.1 and 2.3.2.

Enumerating the plugins list

As mentioned in Section 2.3.1, using the installed add-ons of a browser instance is a very strong and often used feature in Browser-Fingerprinting. Getting this list used to be quite easy. One can just iterate through the `navigator.plugins` array using JavaScript. Listing 2.1 shows, how this can be done in real-life, along with its output.

[5]This web site was visited with a MacBook Pro 13", Early 2011, Firefox 37.0.2, Java, Flash and JavaScript were activated. The test was executed on 07.05.2015.

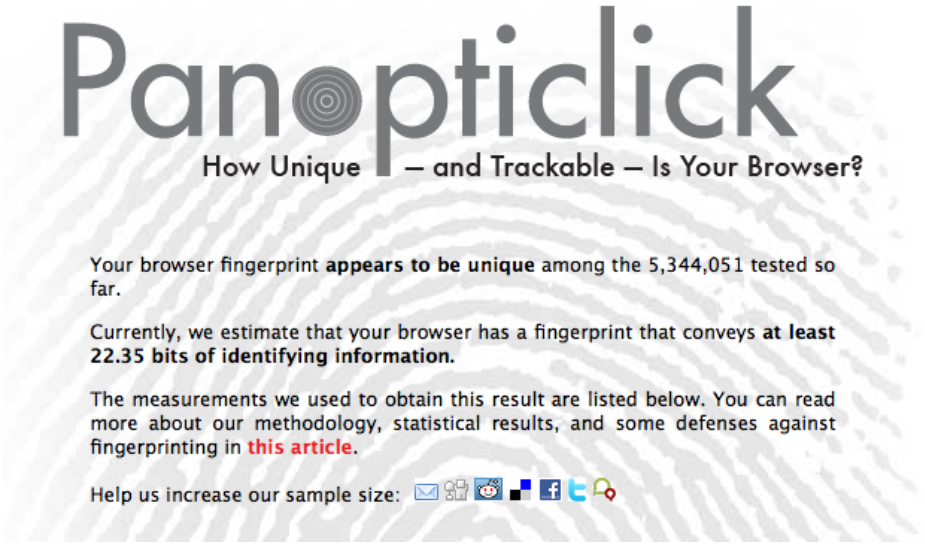

Figure 2.3.: Exemplary result of the Panopticlick project.

Table 2.2.: Entropy of the Features used in the Panopticlick project.

Feature	Entropy (bits)
user_agent	10.0
plugins	15.4
fonts	13.9
video	4.83
supercookies	2.12
http_accept	6.09
timezone	3.04
cookies_enabled	0.353

Listing 2.1: Getting a list of installed browser Add-ons.

```
1  for (plugin in navigator.plugins) {
2    console.log(plugin.name);
3  }
4
5  "Shockwave Flash"
6  "QuickTime Plug-in 7.7.3"
7  "Default Browser Helper"
8  "Unity Player"
9  "Silverlight Plug-In"
10 "Java Applet Plug-in"
11 "Adobe Acrobat NPAPI Plug-in, Version 11.0.02"
```

Browser vendors such as Microsoft or Mozilla are working hard to hamper Browser-Fingerprinting. Since the list (and its order) of the installed browser add-ons is such a strong feature, the vendors were looking for solutions that would make exploiting the plugins list more difficult. Microsoft, for example, forbids to iterate over the `navigator.plugins` array in their browser Internet Explorer. Mozilla did something different starting with Firefox 28 [12]. They implemented a technique called *cloaking.*, which means that iterating over `navigator.plugins` does not yield all installed add-ons, but only the common ones like `Shockwave Flash`, `Quicktime Plug-in` or `Java Applet Plugin-in`. In theory, it is possible to cloak every add-on. The reason for this to be not implemented yet, is compatibility with web sites. A lot of them check for add-ons by iterating the add-on list and comparing each add-on with the looked for add-on. Cloaking all add-ons would result in this check to fail and therefore, the web sites might break [12]. Uncommon add-ons

such as Adblock Plus or any other add-on do not appear in the output. In Listing 2.2, it can be seen, how cloaking add-ons influences the output compared to the output without cloaking.

Listing 2.2: Getting a list of installed Browser Add-ons when cloaking is implemented.

```
for (plugin in navigator.plugins) {
  console.log(plugin.name);
}

"Shockwave Flash"
"QuickTime Plug-in 7.7.3"
"Java Applet Plug-in"
```

While cloaking effectively hampers the use of the `navigator.plugins` array to get a list of all installed add-ons, it is not the ultimate solution to the problem. There is still a mechanism to check, if a specific add-on is installed. One can still explicitly check for a given add-on, for example by doing navigator.plugins[“Silverlight Plugin-In“].

Fingerprintjs

Another interesting project in the field of Browser-Fingerprinting is the open-source fingerprinting library *fingerprintjs* [13] [6]. This library is written purely in JavaScript and aims to create a unique fingerprint of a browser. Table 2.3 shows, which features are part of the fingerprint [15, 16, 13].

Table 2.3.: Features used in fingerprintjs.

Variable	Function
User Agent	User Agent String for the current browser.
Language anguage	Language preferred by the user, usually the language of the browser UI
Color depth	Color depth of the screen.
Screen resolution	The screen resolution of the current monitor used
Timezone difference	Timezone difference between UTC and Local Time
SessionStorage supported?	Is SessionStorage supported?
LocalStorage supported?	Is LocalStorage supported?
database type	HTML5 offers local databases. If those are used, their type is also part of the fingerprint.
CPU class	CPU type
Platform	The platform of the browser.
Do Not Track	Represents the user’s do not track preferences.
Browser add-ons	Get a list of the browser add-ons. There is some special handling for Internet Explorer.
Canvas-Fingerprint	Use the HTML5 canvas element to create a fingerprint. This part will be described in more detail in Section 2.3.2.

Using this library is plain simple, as Listing 2.3 shows.

Listing 2.3: Using fingerprintjs to create a fingerprint.

```
// Creating a fingerprint
var fingerprintWithoutCanvas = new Fingerprint().get();

// Creating a fingerprint by adding Canvas-Fingerprinting additionally.
var fingerprintWithCanvas = new Fingerprint({canvas: true}).get();
```

[6] Please note that there exists a successor: `fingerprintjs2` [14]. This library is currently under development and not ready for productive usage. Moreover, it has been released at a time, when this thesis already focused on fingerprintjs, which will be focused on in this thesis.

```
6
7  console.log("Fingerprint without Canvas-Fingerprinting: " + fingerprintWithoutCanvas);
8  console.log("Fingerprint with Canvas-Fingerprinting: " + fingerprintWithCanvas);
9
10 // OUTPUT:
11 // Fingerprint without Canvas"=Fingerprinting: 3343837964
12 // Fingerprint with Canvas"=Fingerprinting: 743329353
```

As one can see in the output, the `fingerprintjs` library hashes the fingerprint using (by default) a hash function called MurmurHash [17]. This hash function's output is always a 32bit integer number, as one can see in the example output in Listing 2.3. The Canvas-Fingerprinting part is realized by printing the text "http://valve.github.io" on a canvas and extracting the produced pixels. In-depth knowledge about that technique is presented in Section 2.3.2

This Section gave an overview of Browser-Fingerprinting, a tracking mechanism capable of fingerprinting machines. It has been depicted that its ability to identify browsers is impressively accurate, even without using a great number of features. With *fingerprintjs*, an open-source library implementing Browser-Fingerprinting has been presented. Due to this library, tracking users with fingerprinting techniques is extremely simple and yet powerful.

2.3.2. Canvas-Fingerprinting

Canvas-Fingerprinting is a relatively new technique that can be considered a special variant of Browser-Fingerprinting. This Section will give an overview of Canvas-Fingerprinting and explain the idea behind it. Moreover, the single parts of this technique are described, starting with the most general form: rendering a text on a `<canvas>` element. Subsequently, the mechanisms behind WebFonts and how they can be used to find out the list of installed fonts on a machine will be described. The last part of this Section will focus on WebGL, another possibility to implement Canvas-Fingerprinting.

General Information

Browsers are becoming more and more complex. To tackle future challenges regarding performance or battery life, browsers and the underlying operating system are tied together closely. This results in web sites having access to resources of the operating system. One of the mainspring behind this development is the HTML5 suite of specifications [18], which provides - among others - new functionalities like embedding sound or video in web sites, a client-side datastore, geolocation services, a 3D graphics (WebGL) or a drawing surface that supports drawing programmatically (the `<canvas>` element). Especially the last one utilizes resources from the operating system, because using the graphic card for drawing provides huge performance advantages. As a result of these access possibilities the browser behavior depends on the resources of the operating system.

Canvas-Fingerprinting exploits this tight relation between operating system and browser. It describes a new type of fingerprint based on browser font and WebGL rendering. The fingerprint is created by rendering text and/or WebGL scenes to a `<canvas>` element. After that, the pixels produced are taken into account. Those pixels show minimal differences, depending on the graphics card and its driver. Even very simple drawings - for instance rendering a simple sentence in a very common system font - provide huge differences. In the following, fingerprints created by using Canvas-Fingerprinting will be called *canvas fingerprints*. A canvas fingerprint has the following properties [19]:

1. *Consistency*: The canvas fingerprint is consistent, meaning that the produced pixels are identical during several executions for one user.
2. *High entropy*: The fingerprint provides a lot of information, which makes it possible to distinguish the machines.
3. *Independence*: Due to the fact that Canvas-Fingerprinting relies on the graphics card and its driver, the canvas fingerprint is independent of other fingerprints mentioned in 2.3.1.

4. *Transparency*: Creating the canvas fingerprint is performed in background and therefore not noticed by the user. Moreover, its execution time is only a fraction of a second and consequently not measurable.

5. *Availability*: Using Canvas-Fingerprinting only requires JavaScript to be activated and the `<canvas>` element to be supported. Both conditions are almost aways true on modern browsers.

HTML5 Canvas

With the HTML5 `<canvas>` element, it is possible to draw programmatically in the browser. In fact, this is really simple to apply. All one has to do is getting a graphics context by calling the `getContext('2d')`[7] function on the canvas object and use its methods and properties to draw. There exist a plethora of methods and properties, some of them are shown in Table 2.4 [20].

Table 2.4.: Selection of methods and properties of the 2D context.

Type	Name	Description
Property	fillStyle	Sets or returns the color, gradient, or pattern used to fill the drawing
Property	strokeStyle	Sets or returns the color, gradient, or pattern used for strokes
Method	fillRect	Draws a "filled" rectangle
Method	strokeRect	Draws a rectangle (no fill)
Method	lineTo	Adds a new point and creates a line from that point to the last specified point in the canvas
Property	textBaseline	Sets or returns the current text baseline used when drawing text
Property	font	Sets or returns the current font properties for text content
Method	fillText	Draws "filled" text on the canvas
Method	strokeText	Draws text on the canvas (no fill)
Method	toDataURL	Returns a Base64 encoding of a PNG image that involves the whole contents of the canvas.

Listing 2.4 gives an example, how Canvas-Fingerprinting is used in the wild. In fact, this code snippet stems from `http://www.pof.de`, a web site that is using both fingerprinting techniques extensively.

Listing 2.4: Using Canvas-Fingerprinting.

```
1  var canvas = document.createElement('canvas');
2  var ctx = canvas.getContext('2d');
3
4  var txt = 'http://www.plentyoffish.com';
5  ctx.textBaseline = "top";
6  ctx.font = "14px 'Arial'";
7  ctx.textBaseline = "alphabetic";
8  ctx.fillStyle = "#f60";
9  ctx.fillRect(125,1,62,20);
10 ctx.fillStyle = "#069";
11 ctx.fillText(txt, 2, 15);
12 ctx.fillStyle = "rgba(102, 204, 0, 0.7)";
13 ctx.fillText(txt, 4, 17);
14
15 var data = canvas.toDataURL()
```

[7] In the current HTML5 specification, the 2D context is the only one defined [19].

Figure 2.4.: The picture produced by the source code in Listing 2.4.

At first, a `<canvas>` element is created. However, this element does not provide support for drawing on its own. Instead, it is necessary to get a `context` object. Between line 4-7 in Listing 2.4, some traits are defined, such as the text to be drawn, its size, font and location. Between line 5-13, text and rectangles are painted (along with some trait-defining like in line 10 and 12). By calling the `toDataURL` method on the `<canvas>` element, one can extract the information of the drawn picture on a pixel-level. This means that every pixel is taken into account in the return value of this function. This is the reason, why Canvas-Fingerprinting works. Please note that the `<canvas>` element is not appended to the body and therefore, not displayed on the web page. This is not a mistake but done on purpose, because displaying the drawn image is not relevant. It is absolutely enough, the extract the information of the drawn image and use them as a fingerprint (or at least as a part of a fingerprint). The resulting image of the canvas drawing is shown in Figure 2.4.

According to [19], there exist two different areas of application for canvas fingerprints: black box and white box.

- *Black box*: The `<canvas>` element can be used to create a distinguishable fingerprint, which is consistent, as long as hardware and software are unchanged. Since there is no information given about the implementation, this is called black box.
- *White box*: On the other hand, the `<canvas>` element could be used to draw conclusions about the hardware, software and the configuration of a system. One could match the data URL produced by the `toDataURL` method with a set of data URLs, whose hardware, software and system properties are known. As a result, it might be possible to reveal private information about the user's system. Furthermore, the gathered information might be suitable to improve attacks on the system.

WebFonts

WebFonts is another mechanism which can be utilized for Canvas-Fingerprinting. WebFonts are specified in CSS3 and enable web developers to load a font from another server. This circumvents the problem that web developers are dependent on the system fonts installed on the user's computer. To download a font, the web developer adds a `@font-face` CSS rule along with a `src` attribute that references the location of the font. It is than the browser's job to download this font and to make it available for the web site. Moreover, web fonts can be used when drawing on `<canvas>` elements.

Another interesting aspect of WebFonts is the browsers so called fallback strategy. When specifying an external font, the browser must decide, what to do, if the font can not be loaded or found. In most cases, browsers use a predefined font as fallback. This behavior leads to another very worthwhile mechanism. Fingerprinters can exploit the fallback mechanism to detect installed fonts, which can in turn be used as a part of a fingerprint. As shown in [21], it is sufficient to use a combination of JavaScript and CSS to be able to detect fonts. The reason for this to work is the simple fact that each character is represented differently in each font. As a consequence, the same string's overall width and height will be different in almost every font.

The test for fonts is then done as follows: First of all, a given string is created with a predefined size and specified font. Then, its height and width are saved as reference values. After that, the same text is rendered having the searched font and the same size. The height and width of the resulting are also read out. Due to the fallback mechanism, the fallback font is taken if the searched font could not be found. As a consequence, if the height and width of the new font is equal to the height and width of the fallback font, the searched one is not present on the computer. The library presented in [21] performs this test with three different standard fonts and provides a success rate of almost 100%. Listing 2.5 shows the basic principles.

Listing 2.5: Font-Detection using JavaScript and CSS.

```
1 // Standard font. Its height and width are taken as a reference.
2 <span style="font-family: monospace; font-size: 72px">mmmmmmmmmmlli</span>
3
4 // New-Font is the font that is looked for. If this font is not present on the
5 // computer, the next font specified is taken as a fallback (monospace in this case)
6 // The height and width of this <span> element can be compared to the height and width
```

```
// of the first <span> element. If they are equal, the font is not present on the computer
// because the fallback font has been taken.
<span style="font-family: New-Font, monospace; font-size: 72px">mmmmmmmmmmlli</span>
```

By exploiting the fallback mechanism of WebFonts, a tracker is capable of getting to know the exact list of installed fonts on the machine, which is, as Eckersley [11] demonstrated, another very strong part of a possible fingerprint.

WebGL

With *WebGL* [22], there is a JavaScript API that offers rendering 3D graphics in a `<canvas>` element. WebGL is based on OpenGL ES 2.0 and nowadays, all popular (Chrome, Firefox, Safari, Opera) do support this API. Just like the common `<canvas>` element, the contents are rendered using the installed graphics card. Consequently, the WebGL API can also be used for creating a canvas fingerprint. Figure 2.5 shows, what WebGL is capable of. It shows, what Mowery and Shacham [19] has been rendered in the webgl test.

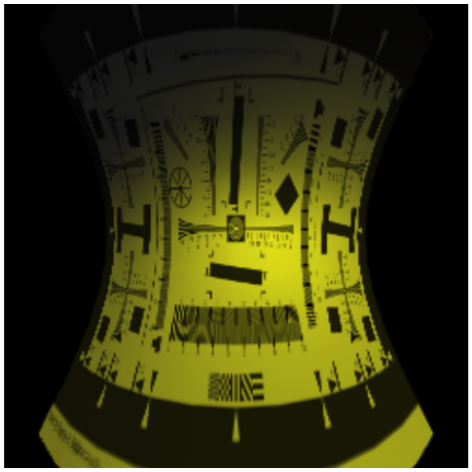

Figure 2.5.: Example of a WebGL rendering.

This Section showed the basics behind Canvas-Fingerprinting, followed by different possible ways of implementation which have been depicted in great detail. It became clear, that Canvas-Fingerprinting can be another possible part of a fingerprint through its high entropy. The next Section will present existing approaches to detect fingerprinting techniques.

2.4. Detection Mechanisms

The last Section's focus lied on Web-Tracking and how it can be deployed. Since this thesis aims to detect Web-Tracking, it is necessary to present detection mechanisms used in areas like Data Mining or Knowledge Discovery in Databases. At first, Section 2.4.1 will give some general information about Data Mining and why it is so important nowadays. Moreover, it will be explained, what classification means in this context. For this thesis, three different techniques of this field are relevant. Bayesian classifiers, Decision Trees and ExtraTrees. Sections 2.4.2, 2.4.3 and 2.4.4 will demonstrate those techniques, as well as highlighting their advantages and disadvantages.

2.4.1. Introduction

Big Data might be one of the most important trends in businesses right now. Since computers become more and more efficient and storage prices are dropping, almost every company uses this development to collect

as many data as possible. Big Data is used in several different areas like health, human resources and even sports [23, 24, 25]. However, the growing amount of data is only useful, if companies manage to extract useful information out of it. Therefore, data specialists are needed to analyze the data. Without that process, Big Data is just a "big trash dump" [26]. This is where Data Mining and Knowledge Discovery in Databases come into play. These disciplines focus on analyzing huge amounts of data, try to gain as much useful information as possible and unsurprisingly experience a lot of growth and recogniction.

This thesis will concentrate on classification, which is a form of data analysis. In classification, models, also called *classifiers*, are created and used to predict *classes*, which can be different in each task. Just consider spam filtering in modern E-Mail clients. The classes, the user is interested in, are *spam* or *no spam*. Since this thesis focuses on Web-Tracking, the classes of importance are *tracking* and *not-tracking*.
Contrary to *classifiers*, there exists another model called *predictor*, which are used in *numeric prediction*. They differ from classifiers insofar as they do not predict a class but a *continuous-valued function* or *ordered values* [27].

According to Han et al. [27], data classification is a two-step-process, with the first step being called *learning step* and the second step being called *classification step*. Those two steps are visualized in Figure 2.6.

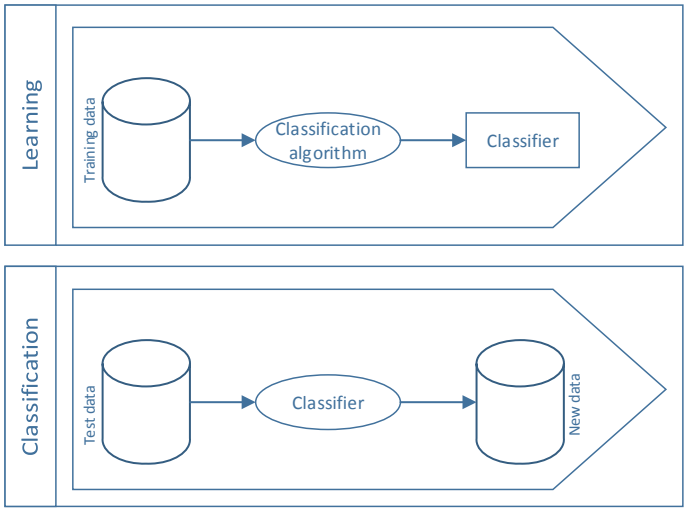

Figure 2.6.: The steps that build the basis of data classification.

- *Learning*: In this step, the classifier is built by using an algorithm that analyzes the tuples of *training data*. Since each tuple's class is already predefined, this procedure is called *supervised learning*[8]. There exist several variants of how a classifier is represented, for example in form of classification rules or a Decision Tree.

- *Classification*: After the classifier has been created, it has to be tested. It is measured, how well the classifier performs in terms of accuracy. It is important to use a different set of data for the test runs, because testing with the same data might lead to overly precise results, which don't reflect the actual accuracy. This is due to *overfitting*, which describes the problem that a classifier might adapt to peculiarities present in the training set, but not present in the general data set. If the measured accuracy of the independent test data is acceptable, the classifier can be used to classify unknown data.

Throughout this work, the following nomenclature will be used. The complete data available is called *training data*. It consist of numerous tuples. Each of those tuples is denoted as X. X is an n-dimensional feature vector, where each part of X represents one *feature* and its assigned class. The feature used for a split is called *splitting feature*.

[8] As expected, there also exists *unsupervised learning*. In this procedure, the classes of the tuples in the training set are not known before and even the total number of classes that will have to be learned, is unknown.

2.4.2. Bayesian Classifiers

Bayesian classifiers are based on statistical techniques. They are capable of predicting, if a given tuple belongs to a specific class or not. Bayesian classifiers are based on the *Bayes' theorem*, which has been stated by Thomas Bayes, an English statistician, who worked in the field of probabilities [28]. He formulated the theorem as shown in Equation 2.1.

$$P(H|X) = \frac{P(X|H) \cdot P(H)}{P(X)} \tag{2.1}$$

The single parts of this equation are defined as follows [27]:

- $P(H|X)$ is called *posterior probability* of X conditional on H. This is the probability which will be calculated.
- $P(X|H)$, $P(H)$ and $P(X)$ are *prior probabilities*. Those can be computed by using the training data.

Bayes classifiers are known to have a high performance and accuracy [27]. *Naïve Bayesian Classifiers*, which is a simple Bayes classifier, is on par with Decision Trees and neutral network classifiers regarding speed and performance. Moreover, Bayes classifiers are tested and proven to work. One of their main area of application is spam filtering in E-Mails. In 2002, Paul Graham published an article called "A Plan for Spam" [29], in which he suggested to use an approach based on Bayes' theorem to filter spam in E-Mails. This suggestion was a huge success, resulting in many software developers implementing Bayesian classifiers in their E-Mail clients. In fact, even Thunderbird[9], one of the most popular open-source E-Mail clients, implements Bayesian classifiers to filter out spam [30].

Naïve Bayesian Classifiers

Naïve Bayesian classifiers are strictly speaking a simple form of Bayesian classifiers. They have special properties, the most important ones for this thesis are listed below [27].

- As mentioned in 2.4.1, the training data consist of tuples X, which are composed of n features.
- Moreover, it will be assumed that there are m classes, $C_1, C_2, \ldots, C_m$. The conditional probability for a tuple X to belong to a class C_i is computed as shown in Equation 2.2:

  $$P(C_i|X) = \frac{P(X|C_i) \cdot P(C_i)}{P(X)} \tag{2.2}$$

  If there exists more than one class, one has to calculate the conditional probability for every single one of them. The class with the highest probability conditional on X will then be the predicted class of X. This means that the probability $P(C_i|X)$ has to be maximized over all tuples X.
  To sum it up, the Naïve Bayesian classifier will predict that a tuple X belongs to the class C_i with the highest probability conditioned on X.
- As it becomes clear, the denominator of all these calculations mentioned in the last point is always the same: $P(X)$. Thus, it can be ignored and it's sufficient to maximize the numerator $P(X|C_i) \cdot P(C_i)$. In case of a prior probability of a class being unknown, one will assume that each class is equally likely. Consequently, it would be enough to maximize $P(X|C_i)$.
- As mentioned before, X is a tuple with several features. The number of those features can be quite high, which would mean that the calculation of $P(X|C_i)$ would be extremely expensive. Here, the "naïve" part comes into play, meaning that for Naïve Bayesian classifiers, it will be assumed that each feature of X is conditionally independent of one another. Of course, this assumption is not always correct. Considering the previous spam filter example, it seems pretty clear that for example the occurrence of the word "Württemberg" is not independent of the word "Baden", since those two are almost always used

[9] https://www.mozilla.org/de/thunderbird/

together. However, using the independence assumption, $P(X|C_i)$ can be calculated as demonstrated in Equation 2.3:

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = \tag{2.3}$$

$$P(x_1|C_i) \cdot P(x_2|C_i) \cdot P(x_3|C_i) \cdot \ldots \cdot P(x_n|C_i) \tag{2.4}$$

The probability $P(x_k|C_i)$ denotes the probability of the feature x_k in all tuples with the class C_i and is easy to estimate using the training set.

Bayesian classifiers are great regarding performance and accuracy. Yet, since they assume (single) features to be conditionally independent, they also provide some inaccuracies.

In order to make things clearer, a simple example will be introduced at this point. Let's assume, Table 2.5 represents the training data[10].

Table 2.5.: Exemplary training data for the Naïve Bayes example.

Weather	Depth of snow	Skiing?
Sunshine	< 50	no
Rain	< 50	no
Rain	≥ 50	no
Snow	≥ 50	yes
Snow	< 50	no
Sunshine	≥ 50	yes
Snow	≥ 50	yes
Rain	< 50	yes

The prior probabilities would then be calculated using the formula depicted in Equation 2.2. Those probabilities depend completely on the training set and computing them can be seen as a training of the classifier.

Table 2.6.: Prior probabilities for the training data.

		Weather			Snow	
	prior	Sunshine	Snow	Rain	≥ 50	< 50
$Skiing$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{2}{4}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{4}$
$\neg Skiing$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{2}{4}$	$\frac{1}{4}$	$\frac{3}{4}$

Given these prior probabilities, it is now possible to determine the posterior probabilities using the assumption of conditional independence. Let's assume $X = (Weather = Sunshine, Snow \geq 50)$ to be a tuple, whose class should be predicted using this classifier. By calculating the posterior probabilities for $Skiing$ and $\neg Skiing$, one can determine its class.

[10]This example is taken from exercise eight of the course "Knowledge Discovery in Databases", held at the Ludwig-Maximilians-University in SS2014 [31].

$$P(Skiing|Weather = Sunshine, Snow \geq 50) =$$
$$= \frac{P(Weather = Sunshine|Skiing) \cdot P(Snow \geq 50|Skiing) \cdot P(Skiing)}{P(Weather = Sunshine, Snow \geq 50)}$$
$$= \frac{\frac{1}{4} \cdot \frac{3}{4} \cdot \frac{1}{2}}{P(Weather = Sunshine, Snow \geq 50)} =$$
$$= \frac{\frac{3}{32}}{P(Weather = Sunshine, Snow \geq 50)}$$

$$P(\neg Skiing|Weather = Sunshine, Snow \geq 50) =$$
$$= \frac{P(Weather = Sunshine|\neg Skiing) \cdot P(Snow \geq 50|\neg Skiing) \cdot P(\neg Skiing)}{P(Weather = Sunshine, Snow \geq 50)}$$
$$= \frac{\frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{2}}{P(Weather = Sunshine, Snow \geq 50)} =$$
$$= \frac{\frac{1}{32}}{P(Weather = Sunshine, Snow \geq 50)}$$

Since $P(Weather = Sunshine, Snow \geq 50)$ is the same in both cases, it can be ignored. Moreover, as stated before, the assigned class is the one with the highest posterior probability. Clearly, $\frac{3}{32} > \frac{1}{32}$ hence the class for X would be $Skiing$.

2.4.3. Decision Trees

A Decision Tree is a flowchart-like structure with the following components [27]:

- Internal node: Represents a test on a feature, usually denoted as a rectangle.
- Branch: Represents the outcome of the test.
- Leaf node: Represents the class. It is depicted by a oval.
- Root node. The topmost node that has only leaving, no incoming nodes.

Training Decision Trees requires the same data structures that have already been introduced: tuples X, consisting of n features. Right now, there exist several algorithms to create a Decision Tree: *ID3*, *C4.5*, *C5.0* and *CART*. ID3 is the oldest among them, being developed in 1986 by Ross Quinlan. C4.5 followed ID3, providing some improvements like removing the restriction that features must provide categorical data. C5.0 represents another improvement of C4.5. It was also developed by Quinlan. C5.0 is for example more accurate than C4.5. The last algorithm mentioned here is CART (Classification and Regression Trees) which has several similarities to C4.5. Unlike C4.5, it is capable of numerical target variables and it does not compute rule sets [32].

Constructing Decision Trees follows some basic procedures, as shown below [33].

1. At the beginning, the complete training data is assigned to the root node.
2. The next *splitting feature* is selected. This is done using the so called *splitting strategy*. Splitting strategies will also be handled topic in this Section.
3. The training data is being partitioned using the splitting feature.
4. This procedure is recursively applied to the resulting partitions.
5. Break conditions:
    - There are no more splitting features available.

- All training data of a node belong to the same class.

Decision Trees are very popular, because they have several advantages. Some of them are listed below [32]:

- They can be visualized, which makes understanding and interpreting them a lot easier.
- Training a Decision Tree is comparatively convenient, since the training data does usually not need to be preprocessed, for instance by removing blank values.
- Predicting data is very cheap, because only the trees height is relevant, which is usually logarithmic to the number of training data.
- Decision Trees can process both, numerical and categorical data.
- The performance of Decision Trees is usually good.

Nevertheless, they are not perfect and also have some disadvantages:

- Decision Trees are prone to overfitting.
- They are extremely dependent on the training data. Even small changes may result in a completely different tree.
- Creating a optimal Decision Tree is a problem known to be NP-complete. Therefore, algorithms creating Decision Trees are not able to produce the globally optimal tree.
- If the training data is not balanced meaning that some classes occure more dominantly, the algorithms might create biased trees.

Splitting Strategies

The splitting strategy plays a central role when building Decision Trees. The idea is to split the training data at a given node into subsets of different classes. Ideally, all members of a subset have the same class. Hence, the *best* strategy for splitting would be the strategy that comes as close as possible to this idealized scenario of perfect splitting.
At each node, a ranking of all features present in the training data is created. According to that ranking, the best feature for a split is selected. This can either be the one with the highest or lowest score, depending on the strategy. Moreover, one has to distinguish between two kinds of splits: categorical splits or numerical splits. The first ones are splits based on conditions like $feature = a$ or $feature \in set$. Such splits possibly result in a lot of different subsets. Numerical splits however divide the training set according to conditions like $feature < a$. This might lead to a lot of different splitting points [33].

For this thesis, two splitting strategies are relevant: *Information Gain* and *Gini Index*, which are described in the following Sections.

Information Gain This splitting strategy is build on the already mentioned entropy, which has been originally introduced by Claude Shannon, who worked in the field of information theory. In the context of Decision Trees, entropy can be seen as a measure of impurity [27] that reaches its maximum, when there are two classes and each of them has a probability of $\frac{1}{2}$. One classic example for this is a coin toss, where $p(heads) = \frac{1}{2}$ and $p(tails) = \frac{1}{2}$. The entropy of a coin toss is maximal, since it is not possible to predict, what the result will be. However, if the coin is manipulated such that $p(heads) = 1$, the entropy of this experiment would be 0, because there is no additional information provided, when the result of the experiment is always the same.

The *entropy* of a partition D consisting of a set of training data is defined in Equation 2.5 [33]. m denotes the number of partitions, k is the total number of classes.

$$entropy(D) = -\sum_{i=1}^{k} p_i \cdot \log_2(p_i) \tag{2.5}$$

So let's assume, the feature A divided the training data into partitions $D_1, D_2, \ldots, D_m$. The *Information Gain* of A related to D is than defined as follows [33], where $|D_i|$ denotes the number of total tuples in the partition D_i.

$$gain(D, A) = entropy(D) - \sum_{i=1}^{m} \frac{|D_i|}{|D|} \cdot entropy(D_i) \tag{2.6}$$

$gain(D, A)$ compares the entropy before the split with the entropy after the split. Its result can be seen as the increase in information of the split - the Information Gain. Another way to interpret this would be to see it as a reduction in insecurity in the subtree prediction outcome.

For each node of the tree, the Information Gain is calculated for each feature and the one with the highest Information Gain is than selected as splitting feature.

Let's use an example to illustrate this procedure[11]. Assuming, one wants to predict the risk class of car drivers based on the data presented in Table 2.7.

Table 2.7.: Exemplary training data for the Decision Tree example.

Person #	Time since getting the driver license	Gender	Residence	Risk class
1	$1-2$	m	City	low
2	$2-7$	m	Countryside	high
3	>7	w	Countryside	low
4	$1-2$	w	Countryside	high
5	>7	m	Countryside	high
6	$1-2$	m	Countryside	high
7	$2-7$	w	City	low
8	$2-7$	m	City	low

In order to predict the risk class, it is necessary to construct a Decision Tree. In this example, the splitting strategy will be the information gain.

First of all, one has to find a feature to split on. Therefore, it's necessary to compute the entropies of all possible features. After that, one has to calculate their information gain and choose the one with the highest value.

The entropy of the training data, $entropy(D)$, is 1, since $p(riskClass = low) = \frac{1}{2}$ and $p(riskClass = high) = \frac{1}{2}$.

The first feature to examine is *Time*. Using this feature as splitting feature would result in three different partitions: $1-2, 2-7, >7$. In order to be able to calculate the Information Gain of *Time*, it is essential to compute the entropy of the the resulting partitions. The entropy for each partition is calculated below.

[11] Again, this example is taken from the course "Knowledge Discovery in Databases" that was held in SS14 [31].

$$time = 1 - 2$$
$$D_1 = (Person1, Person4, Person6)$$
$$p(RiskClass = low) = \frac{1}{3}$$
$$p(RiskClass = high) = \frac{2}{3}$$
$$entropy(D_1) = -\sum_{i=1,2} p_i \cdot \log_2(p_i) =$$
$$= -(\frac{1}{3} \cdot \log_2(\frac{1}{3}) + \frac{2}{3} \cdot \log_2(\frac{2}{3})) \approx 0.918$$

$$time = 2 - 7$$
$$D_2 = (Person2, Person7, Person8)$$
$$p(RiskClass = low) = \frac{2}{3}$$
$$p(RiskClass = high) = \frac{1}{3}$$
$$entropy(D_2) = entropy(D_1) \approx 0.918$$

$$time = 7$$
$$D_3 = (Person3, Person5)$$
$$p(RiskClass = low) = \frac{1}{2}$$
$$p(RiskClass = high) = \frac{1}{2}$$
$$entropy(D_3) = 1$$

Using these entropies, it is possible to calculate the Information Gain for the feature *Time*.

$$gain(D, time) =$$
$$= entropy(D) - \sum_{i=1}^{m} \frac{|D_i|}{|D|} \cdot entropy(D_i) =$$
$$= 1 - (\frac{3}{8} \cdot 0.918 + \frac{3}{8} \cdot 0.918 + \frac{2}{8} \cdot 1) \approx 0.06$$

To choose the best splitting feature, the exact same calculations also have to be executed for the features *Gender* and *Residence*. While the detailed calculations will be spared out at this point, the results for the Information Gain of both features, *Gender* and *Residence* are given below.

$$gain(D, Gender) \approx 0.05$$
$$gain(D, Residence) \approx 0.55$$

Clearly, the Information Gain of the feature *Residence* has the highest value and is thus taken as a splitting feature. Figure 2.7 shows, how the Decision Tree would be visualized after the first split.

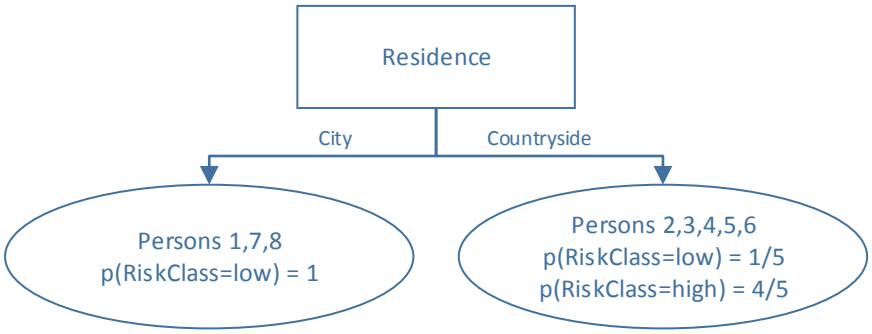

Figure 2.7.: Visualized Decision Tree after the first split.

After this first step, the algorithm would than recursively go on and look for the best splitting feature in the left (Persons 1, 7, 8) and the right (Persons 2, 3, 4, 5, 6) node. Since all persons of the left node already belong to the same class ($p(RiskClass = low) = 1$), there is no split necessary in this node. Figure 2.8 shows the results, after all splits have been done recursively in all nodes.

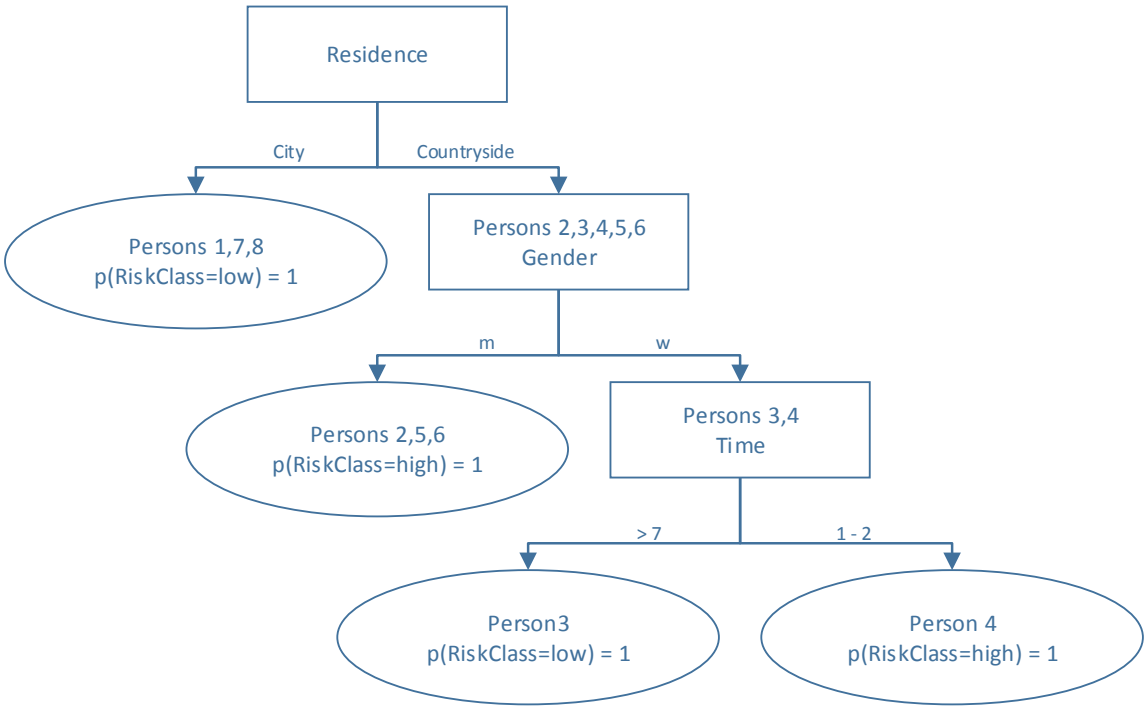

Figure 2.8.: Visualized Decision Tree after every split has been done.

Figure 2.8 also shows that the next feature used as splitting feature was *Gender*, although the actual information gain for *Gender* and *Time* are equal with a score of 0.322. In this case, the selection of the splitting feature is not relevant. This split divided the training data in two sets: $D_1 = (Persons2, 5, 6)$ and $D_2 = (Persons3, 4)$. In D_1, every person has the same class (low) assigned. Therefore, this node becomes a leaf node and it is not necessary to split. Partition D_2 however needs to be split again. The only feature remaining here is *Time*, which is consequently taken. The two resulting partitions consist of $Person3$ with the class low and $Person4$ with the class $high$.

Using this Decision Tree to predict a class of a new test data is relatively simple. One has to walk down the tree beginning at its root node and compare the relevant features at each node. For example, let $A = (Time = 1, 2; Gender = w; Residence = Countryside)$ be a test set, which class has to be predicted. At first, one

starts at the root. The first relevant feature is $Residence$ and the left path would be taken if the residence is $City$, otherwise the right path is taken. Here, the feature to look for is *Gender* and since $Gender = w$, again the right path is chosen. The last relevant feature is *Time*. Because of $Time = 1, 2$, one picks the right child node. This node is a leaf node and holds expectably a class, in this case $high$. So this Decision Tree would predict the class $high$ for this exemplary test data.

This example showed, how a Decision Tree would be built with Information Gain as splitting strategy. However, there are other splitting strategies available, for example *Gini Index*.

Gini Index Another splitting strategy, which is often used when building Decision Trees, for example in the original CART algorithm, is the *Gini Index*.
The Gini Index is a measurement for the impurity of a partition D. It is calculated as follows [27].

$$gini(D) = 1 - \sum_{i=1}^{k} p_i^2 \tag{2.7}$$

p_i is assessed as the number of appearances of class C_i in D (depicted as $|C_{i,D}|$) divided by the total number of tuples in D: $\frac{|C_{i,D}|}{|D|}$.

Assuming that a feature A has led to a division of the training data into partitions $D_1, D_2, \ldots, D_m$, the Gini Index of A in relation to D is than defined as showed in Equation 2.8:

$$gini_A(D) = \sum_{i=1}^{m} \frac{|D_i|}{|D|} \cdot gini(D_i) \tag{2.8}$$

Just like it has been the case with *Information Gain*, the feature with the highest Gini Index will be selected as splitting feature.

Nevertheless, the way Decision Trees are created does not depend on the splitting strategy. This means that the strategy is actually an exchangeable part, only responsible to pick the best splitting feature. The example shown above would be executed the same way, with the only difference being the splitting strategy, which might result in different splitting features at some point.

Tree Pruning

After the tree has been built, a lot of unnecessary branches might have been created, because of possible peculiarities present in the training data resulting in overfitting. By using Tree Pruning, this problem can be tackled. It typically uses statistical measures to remove branches that are not reliable, hence leading to smaller and less complex trees which are easier to understand and to interpret. Moreover, Tree Pruning enhances the performance and the accuracy. Of course, the very simple example presented shown above does not have unnecessary branches due to the small training data. However, pruning is an important part in Decision Trees. A distinction is made between two different pruning approaches [27]:

- *Prepruning:*
  Prepruning is done during the creation of a tree. One could, for example, decide that the training data should not be partitioned any more at a given node. Usually, this is realized by specifying a threshold value. Partitioning the training data, the quality of the resulting split is measured with a given technique like Information Gain or Gini Index. If the quality of the split is lower than the threshold value, the split is aborted and there is no further partitioning allowed. However, choosing the right threshold is a very complex calibration task, since low thresholds might result in only a few branches to be missed out and a high threshold can lead to oversimplified trees.
- *Postpruning:*
  Postpruning is the more common approach which is applied on *fully-grown* trees. It removes complete

subtrees from a tree. The idea is to remove a subtree with all its branches and replace it with a leaf node marked with the most frequent class present in the original branches.

2.4.4. ExtraTrees

With Decision Trees in Section 2.4.3, a tree-based classifier has already been introduced. Decision Trees are great in terms of performance and complexity and their visualization helps to comprehend the classification process. Nevertheless, they also do suffer from some disadvantages, most notably their high variance. Trees are heavily influenced by the randomization of the training data. In fact, each splitting feature and each splitting node is selected based on the training data. This can lead to some problems, when the training data is not structured the right way. Decision Trees are very simple classifiers. In order to have a more complex, but also tree-based classifier the *ExtraTrees* has been picked.

The *ExtraTrees* classifiers has been presented in 2006 in "Extremely randomized trees" [34]. The idea behind it is to tackle the randomization present in the training data by not just building one tree from it, but a lot of trees. Each of those trees are built by selecting the splitting feature randomly. In each node, a random choice of a number of features is picked and among those, the best splitting feature will be selected. Carrying this process to extremes, one can also pick the splitting feature and the cut-points[12] completely randomly and consequently build *totally randomized trees* [34].

Each tree that is part of the ExtraTrees, is built by using the complete training data. The prediction of ExtraTrees is than calculated by aggregating the results of each tree.

This Chapter elucidated the fundamentals required for this thesis, including Web-Tracking and detection mechanisms. The focus concerning Web-Tracking lied on fingerprinting techniques, specifically on Browser- and Canvas-Fingerprinting. It became clear, that those techniques are capable of superseding cookies, which have been the go to mechanism for implementing Web-Tracking until now. Moreover, it has been stated, that fingerprinting techniques are performed in the background, unnoticed from the user. Furthermore, three different classifiers from the fields of Data Mining and Knowledge Discovery in Databases have been depicted: Bayes classifiers, Decision Trees and ExtraTrees.
The next Chapter will now present the current state of research in the according areas.

[12] The cut-points represent nodes used for splitting. Cut-points are only relevant in continuous features. Since in this thesis, there are only categorical features, this is not that important here.

3. Related Work

With the last Chapter introducing the necessary fundamentals concerning Web-Tracking and detection mechanisms, this Chapter will focus on the current status in the research community. Section 3.1 will concentrate on publishments concerning fingerprinting techniques. This part will contain current developments in the field of Browser-Fingerprinting, as well as Canvas-Fingerprinting. Section 3.2 will show the existing ideas to detect Web-Tracking. It will be elaborated that the idea to use traditional techniques from Data Mining like Bayes or Decision Trees for the detection of Web-Tracking is new and that this has not been done before. Lastly, Section 3.3 will focus on counter-measures that can be used to defend himself against fingerprinting attacks.

3.1. Fingerprinting

This Section will concentrate on related work concerning fingerprinting techniques. Section 3.1.1 will cover Browser-Fingerprinting and it will be shown that it can be realized by using not only JavaScript, but also Flash and Java, which can play significant role in this technique. Moreover, current fingerprinting scripts in the wild and the original study of Browser-Fingerprinting [11] are going to be compared. After that, the focus will lie on font enumeration, since this is one of the strongest features available in all browsers. Section 3.1.2 will describe the state of research concerning Canvas-Fingerprinting. It will become clear that the reason for it to work is because of subtle differences in the rendering processes.

3.1.1. Browser-Fingerprinting

Device fingerprinting has been a research topic for a very long time and there exists several different approaches. Mowery et al. [35] proposed techniques to fingerprint a browser based on the JavaScript Interpreter. They revealed that JavaScript performance depends on the browser. By measuring the execution time of instruction sequences and comparing them to the execution time of the same instruction sequences of another browser, they were able to determine the browser family in about 98.2%. A downside of this approach is, however, that it took about three minutes for the execution to finish, which is not feasible in real-life scenarios [35].

Although Eckersley [11] was the first to examine Browser-Fingerprinting on a large scale and drawing lots of attention towards the topic, there has been other research focusing on that area, most notably Jonathan C. Mayer. In 2009, he presented an experiment, in which he fingerprinted 1328 clients [36]. As an identification, he used the concatenated values of `navigator`, `screen`, `navigator.plugins` and `navigator.mimeTypes` and hashed the resulting string. This enabled him to uniquely identify about 96% of the browsers. A year later, Peter Eckersley extended this study on a larger scale and fingerprinted about half a million browsers (see Section 2.3.1).

Eckersley [11] drew a lot attention and, because of that, Browser-Fingerprinting became the research subject of more and more people. Nikiforakis et al. [37] decided in their paper “Cookieless Monster: Exploring the Ecosystem of Web-based Device Fingerprinting” [37] to examine the code of three popular Browser-Fingerprinting providers. As a result, they revealed the techniques that enable web sites to track their users without client-side identifiers like cookies. They also showed that there exist several questionable mechanisms among these techniques. Moreover, they demonstrated how excellent fingerprinting of web browsers work and they explained its reasons. Finally, Nikiforakis et al. [37] investigated browser add-ons that make user-agent spoofing possible and they constituted that these add-ons might even enhance the fingerprintable surface of the browser.

Nikiforakis et al. [37] examined three fingerprinting libraries of large, commercial companies: BlueCava[1], Iovation[2] and ThreatMetrix[3]. Those three were chosen, because two of them appeared in the web tracking survey "Third-Party Web Tracking: Policy and Technology" [38] published by Mayer and Mitchell and the last one was picked because it was highly ranked in a popular search engine.

Table 3.1.: Overview of features used in Panopticlick and in the examined fingerprinting providers.

Fingerprinting Category	Panopticlick	BlueCava	Iovation	ThreatMatrix
Browser customizations	Plugin enumeration$_{JS}$ Mime-type enumeration$_{JS}$ ActiveX + 8 CLSIDs$_{JS}$	Plugin enumeration$_{JS}$ **ActiveX + 53 CLSIDs$_{JS}$** **Google Gears Detection$_{JS}$**		Plugin enumeration$_{JS}$ Mime-type enumeration$_{JS}$ **Flash Manufacturer$_{FLASH}$**
Browser-level user configurations	Cookies enabled$_{HTTP}$ Timezone$_{JS}$ Flash enabled$_{JS}$	**System/Browser/ User Language$_{JS}$** Timezone$_{JS}$ Flash enabled$_{JS}$ **Do-Not-Track User Choice$_{JS}$** **MSIE Security Policy$_{JS}$**	**Browser Language$_{HTTP, JS}$** Timezone$_{JS}$ Flash enabled$_{JS}$ **Date and time$_{JS}$** **Proxy Detection$_{FLASH}$**	**Browser Language$_{FLASH}$** Timezone$_{JS, FLASH}$ Flash enabled$_{JS}$ **Proxy Detection$_{FLASH}$**
Browser family & version	User-agent$_{JS}$ ACCEPT-Header$_{HTTP}$, Partial S.Cookie test$_{JS}$	**User-agent$_{JS}$** **Math constants$_{JS}$** **AJAX Implementations$_{JS}$**	**User-agent$_{HTTP, JS}$**	**User-agent$_{JS}$**
Operating System & Applications	User-agent$_{HTTP}$ Font Detection$_{FLASH, JAVA}$	User-agent$_{JS}$ **Font Detection$_{JS, FLASH}$** **Windows Registry$_{SFP}$**	User-agent$_{HTTP, JS}$ **Windows Registry$_{SFP}$** **MSIE Product Key$_{SFP}$**	User-agent$_{JS}$ Font Detection$_{FLASH}$ **OS + Kernel version$_{FLASH}$**
Hardware & Network	Screen Resolution$_{JS}$	Screen Resolution$_{JS}$ **Driver enumeration$_{SFP}$** **IP Address$_{HTTP}$** **TCP/IP Parameters$_{SFP}$**	Screen Resolution$_{JS}$ **Device Identifiers$_{SFP}$** **TCP/IP Parameters$_{SFP}$**	**Screen Resolution$_{JS, FLASH}$**

Table 3.1 shows the result of the study in form of a taxonomy. The taxonomy illustrates all features that can be collected using a fingerprinting library. It involves all features of Panopticlick and the features used by the examined fingerprinting libraries. The features that are printed boldly are, compared to Panopticlick, either significantly enhanced, gathered through a different method or completely new. Nikiforakis et al. [37] designed the taxonomy as a layered system, where the top layer is represented by the browser and all of its fingerprintable features. The further down the layer resides, the more low-level the features become, resulting in hardware and network related fingerprintable data like TCP/IP parameters.

Hereafter, some of the major differences between Panopticlick and the three examined fingerprinting libraries are explained.

One of the most notable disparity is the heavy usage of Adobe Flash. Nikiforakis et al. [37] discovered that companies providing fingerprinting libraries strongly rely on Adobe Flash, despite the fact that Flash has been under fire for a long time because of its poor performance and it not being very stable. With HTML5, there is actually a newer technology which aims to provide lots of functionalities that used to be realized by Flash.

[1] `http://www.bluecava.com`
[2] `http://www.iovation.com`
[3] `http://www.threatmetrix.com`

Despite these circumstances, Flash is still present on a majority of computers. Flash implements certain APIs existing in the browser and callable via JavaScript, but surprisingly, Flash's APIs do not always provide the same results as the browser APIs do. An example for this unexpected behavior is the response of the platform of execution. While the browser API tells "Linux x86_64", Flash's response is, for example, "Linux 3.2.0-26-generic". As one can see, the result of Flash is more detailed, providing information about the full kernel version. This is not only critical for fingerprinting, but has also be viewed from a security point of view, since possible attackers can now improve their attacks with this additional knowledge.
Another API call, whose results differ in case of Flash and browser is the one that provides information about the used screen resolution. In a dual-monitor setup, Flash[4] serves the sum of both monitor widths, but the browser's API only reports the resolution of the monitor hosting the browser window. This behavior enables the tracker to detect, if a multi-monitor setup is utilized.

A further difference between Panopticlick and the other three fingerprinting libraries is the fact that the libraries do not try to behave the same on different platforms, e.g. Firefox, Internet Explorer or Google Chrome. This means that these libraries adjust their behavior. If the platform is for example Internet Explorer, they try to fingerprint features that are specific for the Internet Explorer. Moreover, they have an extra fallback implementation for enumerating add-ons, since the standard way used in Firefox is not possible in Internet Explorer.

In 2013, Acar et al. [39] presented a framework called *FPDetect*. This framework's goal was the identification and analysis of web-based device fingerprinting. It was designed with scalability in mind. It could be used simultaneously with multiple virtual machines, which enabled the authors to perform a large-scale study examining the Alexa Top 1.000.000.
The framework was capable of detecting JavaScript- and Flash-based fingerprinting attempts and consisted of the following parts:

1. *Crawler*
   Two browsers belong to the crawler: PhantomJS[5] and Chromium[6]. PhantomJS is responsible for JavaScript-based and Chromium is used to examine Flash-based fingerprinting. For controlling the browsers and to visit web sites, CasperJS[7] and Selenium[8] are utilized. The authors chose to adapt the native source code of WebKit, which poses the basis of PhantomJS and Chromium. Another possibility would be a browser extension, but modifying the browser's source code and therefore working on such a low level has several advantages:

   - Browser extensions can only log JavaScript-based events.
   - The origin of events can be detected more precisely.
   - JavaScript getter methods and extensions can be blocked or circumvented.

   The modifications of WebKit allowed the authors to log the access of the following browser and device properties, which are suitable for a fingerprint.

   - `navigator.userAgent`, `navigator.appCodeName`, `navigator.product`, `navigator.productSub`, `navigator.vendor`, `navigator.onLine`, `navigator.appVersion`, `navigator.language`, `navigator.plugins`, `navigator.mimeTypes`, `navigator.cookieEnabled()`, `navigator.javaEnabled()`
   - `navigator.plugins` properties like `name`, `fileName`, `description`, `length`
   - `navigator.mimeTypes` properties like `enabledPlugin`, `description`, `suffixes`, `type`
   - `window.screen` properties like `horizontalDPI`, `verticalDPI`, `height`, `width`, `colorDepth`, `pixelDepth`, `availLeft`, `availTop`, `availHeight`, `availWidth`

[4] This is the case in both Linux implementations, Adobe's and Google.
[5] `http://phantomjs.org`
[6] `http://www.chromium.org`
[7] `http://www.casperjs.org`
[8] `http://docs.seleniumhq.org`

- `offsetWidth` and `offsetHeight` properties and the `getBoundingRect` method that belongs to HTML elements
- `CSSFontFace::getFontData` and `CSSFontSelector::getFontData` methods that are used for font loading

2. *Parser*
   This part's responsibility is to extract useful information from the logs produced by the crawler and save them in the database.
3. *Proxy*
   For Flash, the usage of a HTTP-Proxy is necessary. Since this thesis concentrates on JavaScript-based fingerprinting, the Flash-based part is leaved out. The interested reader may be referred to [39] for further information.
4. *Decompiler*
   The decompiler is relevant for Flash-based fingerprinting. Again, this part is not considered in this thesis.
5. *Central Database*
   Although the framework utilizes several virtual machines for crawling web sites, all machines save their findings (JavaScript function calls, HTTP requests and responses, loaded/requested fonts) in a central database that serves for further analysis.

Acar et al. [39] decided to focus in this study on font enumeration techniques. The reasons for this choice are listed in the following.

- As Peter Eckersley has shown, the installed fonts on a machine represent one of the strongest features, having a entropy of 13.9.
- Fonts do not depend on the operating system. Because of that, different browsers on the same device are considered the same.
- The Tor Browser, being famous for implementing counter-measures against fingerprinting, ships with a specific configuration state. In this state, plugins are disabled and therefore not suitable to use as part of a fingerprint. As a consequence, fonts are the next best feature available.

The following part will concentrate on the JavaScript-based font detection mechanism, since JavaScript is the only relevant technology in this work.

Each experiment executed consisted of two parts. At first, FPDetect was used to crawl a list of web sites. The web sites that were marked by the framework, were potential users of fingerprinting attacks. The authors then analyzed the marked web sites manually in order to rule out False-Positives. A measurement for potential fingerprinters was the number of loaded fonts. To remove web sites that loaded a lot of fonts but did not use them for font detection, the calls of the `offsetHeight` and `offsetWidth` methods of the corresponding HTML elements were counted. The findings of the study are presented in Figure 3.1 [39].

Figure 3.1 shows the Alexa Top 1.000.000 being split into parts of 100.000 web sites, according to their rank within Alexa. Each one of those parts is represented by two bars. The darker bar shows the total number of web sites delivering fingerprinting scripts. However, the authors discovered that not all web sites providing fingerprinting scripts do also execute those scripts. This number of presented as the lighter bar.

To sum it up, FPDetective was able to find 303 web sites among the Top 1.000.000 Alexa that used fingerprinting scripts to track their visitors. Those scripts were delivered by 13 different fingerprinting providers and not all of them had been discovered in early studies.

3.1.2. Canvas-Fingerprinting

The most extensive study in the field of Canvas-Fingerprinting and also the basis of Section 2.3.2 is the work "Pixel Perfect: Fingerprinting Canvas in HTML5". In this paper, Mowery and Shacham [19] examined the

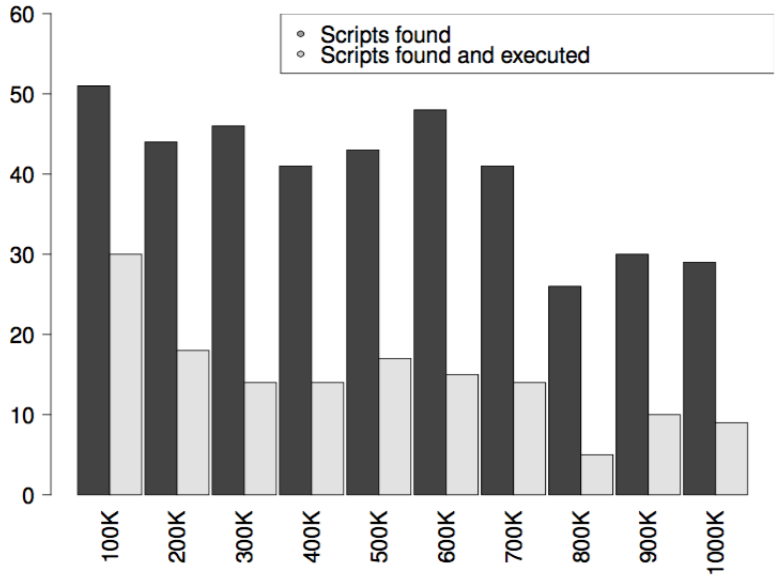

Figure 3.1.: Top 1.000.000 Alexa web sites using JavaScript-based font probing .

suitability of the new HTML5 `<canvas>` element for creating fingerprints. For this purpose, they defined six different tests:

1. *text_arial*
   This was a very basic test with the font being Arial, which is known for its omnipresence in the web. The pangram "How quickly daft jumping zebras vex." was rendered in 18pt to the canvas.

2. *text_arial_px*
   This was basically the same test as *text_arial* with the only difference being the size of the text 20px instead of 18pt.

3. *text_webfont*
   This test differed from the basic *text_arial* test insofar that now, the font was loaded from a web server by using the WebFonts mechanism. The text stayed the same, the size was 12pt and the font used was Stirin Stencil, which was downloaded from the Google Web Fonts server[9].

4. *text_webfont_px*
   Basically the same as *text_webfont*, except for the size being 15px.

5. *text_nonsense*
   From the code point of view, this test was for the most part the same as the Arial tests. However, the font specification was now set to "not even a font spec in the slightest", which was obviously not a valid font and could consequently not be found on the web server. This triggered the fallback strategy, meaning that a default font was taken instead.

6. *webgl*
   This was unsurprisingly the most complex test, since rendering 3D graphics was always a tough task. Basically, the authors rendered some 3D picture on the canvas element. For detailed information about the content of the rendered image, one might refer to [19]. In Figure 2.5 the output produced by an example run of the *webgl* test is presented.

Another point that has to be considered in such tests, is speed. It seems to be pretty clear that fingerprints needing several seconds (or even minutes) to be finished are not feasible. Mowery and Shacham [19] stated in

[9] `http://www.google.com/webfonts`

their paper that all tests were done in a fraction of a second. In fact, the longest delays stemmed from fetching the image assets.

The results of these tests looked very promising. The *text_arial* test revealed 50 distinct versions among the 300 samples collected. In the *text_arial_px* test, the number of distinct versions lowered to 43. This is still pretty impressive, considering the simplicity of this test and the fact that Arial is a font that is 30 years old. Mowery and Shacham [19] came to the conclusion that rendering a simple pangram in a widely used font like Arial is enough to get to know the user's operating system family and, in most cases, the browser family.
The WebFont tests showed similar results. From the 294[10] *text_webfont* tests, there are 45 distinct variants to render the image. The *text_webfont_px* test results were very similar having 44 groups of 294 samples. The authors discovered that some clients did not use the specified font Sirin Stencil but used a fallback font. Four of them (Chrome 16 and 17, Firefox 10 and 11) used Times New Roman and one (Opera 9.8) used a font that seems to be Arial. Nevertheless, since for each of these browser versions existed correct samples in which the specified font was used, it is likely that either WebFonts were disabled on these machines or there occurred an error loading the font. Interestingly, the participants having these results did not improve their anonymity, but in fact, they enhanced their traceability.
The last test concerned WebGL. In 30 test cases, this test delivered no data, which can be ascribed to either WebGL being disabled or a failure occurring in the test. From the remaining 270 successful tests, there were 50 different versions of the rendered image. The authors were quite surprised of the diversity, since the rendered scene only consisted of basic matrix operations. However, they stated that sophisticated WebGL fingerprints might yield better results.

Table 3.2 shows the estimated entropies of the single tests. It is calculated with the same formula presented in Equation 2.5.

Mowery and Shacham [19] highlighted that the 300 samples were not extensive enough to serve as a representative sample of the whole Internet. Therefore, the entropies should be considered rough estimations.

Table 3.2.: Entropies of the Canvas-Fingerprinting tests.

Test	Entropy (in bits)
text_arial	3.05
text_arial_px	2.86
text_webfont	2.93
text_webfont_px	2.95
webgl	4.30
All tests combined	5.73

All single tests combined yielded a entropy of 5.73 bits. Nevertheless, these entropies could be increased by advanced and more specialized tests. The authors had the assumption that it might even be possible to determine the exact installed graphics card, the underlying operating system and the browser family, if targeted tests were created and applied.

While the number of tests in [19] was rather small, Acar et al. [40] performed the first large-scale study in the field of advanced web web tracking mechanisms called "The web never forgets: Persistent tracking mechanisms in the wild". The content of this study was, as the title may already suggest, three different, sophisticated tracking mechanisms: Canvas-Fingerprinting, Evercookies and Cookie Syncing. While Evercookies and Cookie Syncing are very interesting techniques on their own, the focus in this thesis is Canvas-Fingerprinting. Therefore, the concentration lies on this part of the study.

The goal of Acar et al. [40] was - among others - to examine the frequency of usage of Canvas-Fingerprinting. They designed a framework capable of identifying, whether Canvas-Fingerprinting is executed on a web site. Since the focus was on a large-scale, scalability of the framework was always in mind. In order to be able to detect Canvas-Fingerprinting, the authors relied on the findings of Mowery and Shacham [19]. In this publication, Canvas-Fingerprinting was realized by using the `strokeText`, `fillText` and `toDataURL`

[10] In six cases, the specified font could not be loaded due to a race condition in the WebFont Loader library. This library is developed by Google and TypeKit and can be used to request fonts. Moreover, it provides the possibility to register callback functions.

methods. The first two were used to write text on the <canvas> element and the last one extracted pixel-precise data from the <canvas>. In order to detect Canvas-Fingerprinting, the authors opted to modify the browser itself, which was in this case Mozilla Firefox. Since Firefox is a open-source project, its source code can be altered by every interested user. Acar et al. [40] adjusted the source code of Firefox to the effect that the arguments of the strokeText and fillText methods (one part of the arguments is the string that should be drawn to the canvas) and the return value of the toDataURL method were logged. Moreover, they logged the URL of the script that invoked those methods via the nsContentUtils::GetCurrentJSContext method and they co-writed the exact line number (initiator) of this method call via the nsJSUtils::-getCallingLocation. Thus, they were able to determine the fingerprint attempt's script URL and the exact location in the script. The function call logs were evaluated and finally inserted into a SQLite database, which served as basis for further analysis. Notably, the code modifications only contained about 33 lines of code in four different files. The performance implications were only fractional. By using Selenium[11], they visited the top 100.000 Alexa sites using the modified Firefox version and logged the respective function calls for each web site.

Canvas-Fingerprinting is only one way of utilizing the <canvas> element. There are a lot of totally benign reasons to draw text or images. Because of that, the authors had to analyze the data set with a focus on False-Positives. During their examination, they formulated three conditions to filter out False-Positives:

1. Both, the toDataURL and strokeText (or fillText) function calls should be executed and both should be called from the same URL.
2. The overall size of the canvas image(s) should be greater than 16x16 pixels and they should involve more than one color.
3. The requested format of the images should not be a lossy compression format like JPEG.

The first points intention was to remove scripts that use the <canvas> element only for drawing and did not read its content. If Canvas-Fingerprinting is used, drawing only is not enough, the tracker needs to get the content of the <canvas> element in order to create a fingerprint. On the other hand, users who are only using the canvas element for legit reasons like painting a dynamic favicon, usually do not need to extract the canvas content.
<canvas> elements smaller than 16x16 pixels are not suitable for Canvas-Fingerprinting. The reason behind this is the fact that operating systems or font libraries need the fonts to have a minimum size in order to exert anti-aliasing[12].
Lossy compression formats as JPEG are also not fit for Canvas-Fingerprinting, because the minimal differences necessary to create a unique fingerprint may get lost in those formats.

By removing web sites that did not fulfill all of these conditions, Acar et al. [40] managed to lower the number of False-Positives substantially. As a next step, the authors ensured that scripts marked as Canvas-Fingerprinting scripts were also collecting relevant features for the more general Browser-Fingerprinting[13]. Nevertheless, it is not completely safe that no False-Positives are present in the test set, because the above mentioned conditions are not absolutely perfect and there are scenarios, in which web sites did fulfill the conditions but were still not doing Canvas-Fingerprinting. Moreover, a tracker can still perform different tracking mechanisms like pixel stealing by using SVG filters [42] or CSS shaders [43].

Acar et al. [40] came to the conclusion that about 5.5% of the Top Alexa 100.000 sites used Canvas-Fingerprinting. However, it is important to mention that vast majority (about 95%) stemmed from single provider: addthis.com. Still, there were about 20 different domains found, serving Canvas-Fingerprinting scripts and being active on 5542 sites. Eleven providers (and 5532 sites) of the 20 providers were third parties. The authors assumed that these providers did not offer fingerprinting services directly, but delivered fingerprinting as a part of another service. The remaining nine providers (and 10 sites) are fingerprinting scripts that were served from first-parties. It is important to highlight that the crawls only targeted the landing page of a web page. Crawling internal parts of a web sites might result in higher percentage of fingerprinting.
An interesting aspect of the study is the fact that by logging the text that is rendered on the canvas element, the authors identified fingerprintjs being used on a lot of fingerprinting web sites.

[11] Selenium [41] is a test framework for automating browsers. For example, one can control a browser using Selenium and visit web sites.
[12] Anti-aliasing is one of the main reasons for the minimal differences of rendered <canvas> images.
[13] For example, by using static analysis, they checked for features like navigator.plugins or font enumeration.

As a last step, the fingerprinting script of AddThis was analyzed in greater detail, since this script was used for the most part. Interestingly, AddThis's script did not stop at what was published by researchers at this time. The analysis showed that AddThis improved the Canvas-Fingerprinting in order to be able to readout more entropy. The following list shows the additional mechanisms used:

- The fingerprinting script drew the same text twice, using different colors. Moreover, they forced the fallback mechanism by specifying a non-existing font-name starting with *no-real-font-*.
- The text drawn on the canvas was the perfect pangram "Cwm fjordbank glyphs vext quiz".
- The character *U+1F603* was printed on the canvas. By doing so, it was possible to check if Unicode drawing is supported.
- It was inspected, whether canvas *globalCompositeOperation* was supported.
- Two rectangles are drawn and after that, it was checked whether a given point was in the path. This was done by using the *isPointInPath* method.

Finally, it is important to stress out that AddThis reacted against the study presented in this Section. As AddThis stated [44], they deployed the Canvas-Fingerprinting part of their tracking script as a test in order to evaluate the possibility to replace traditional cookies with Canvas-Fingerprinting. They stressed that "the test was completed, the code has been disabled and this data was never used for personalization or targeted advertising" [44]. Moreover, they mentioned that they "don't identify individuals" [44]. Furthermore, they highlighted that data created was only used for internal research and that they acknowledged the opt-out policy during the test.

3.2. Detecting Fingerprinting

This Section represents the work in the field of detecting fingerprinting techniques. Right now, the majority of such tools are provided in form of browser extensions. Section 3.2.1 will present Chameleon, a browser extension for Google Chrome with the aim to detect fingerprinting attempts. Section 3.2.2 will show CanvasBlocker, another extension that is designed to detect Canvas-Fingerprinting.

3.2.1. Chameleon

Chameleon [45] is a browser extension for Google Chrome. Because of the nature of browser extensions, it has only access to JavaScript-based fingerprinting attempts. Chameleon tries to detect fingerprinting by logging calls to browser properties of the `navigator` or `screen` objects. This is done by overwriting the getters of these objects. Chameleon also tries to implement protection, for example by manipulating the properties accordingly. Table 3.3 gives an overview of the fingerprinting techniques that are detected and for which Chameleon provides protection.

As one can see in Table 3.3, Chameleon provides a broad spectrum of fingerprinting-related features. It strikes that a lot of features concerning the enumeration of properties of the `window` and `navigator` objects, are detected and, in some cases, even protected against. Moreover, font enumeration, a pretty strong feature, is also detected but not yet protected against. Nevertheless, as mentioned before, due to the nature of browser extensions, detecting Flash-based fingerprinting attempts is not possible. This means that a potent attack vector is ignored completely. However, this is the price one has to pay when using browser extensions.

3.2.2. CanvasBlocker

CanvasBlocker [46] is another extension with the focus on detecting Canvas-Fingerprinting. Its basic functionalities will be presented in Section 3.3.2. Unlike Chameleon, it leaves out most of the significant features of Browser-Fingerprinting. In order to detect and block it, CanvasBlocker concentrates on the `<canvas>` element.

Table 3.3.: Functionalities of Chameleon.

Fingerprinting technique	Detection	Protection
Request header values	✓	✗
window.navigator values	✓	✓
window.navigator enumeration	✓	✗
window.screen values	✓	✓
Date/time queries	✓	✓
Font enumeration	✓	✗
System color enumeration	✗	✗
CSS media queries	✗	✗
Canvas image data extraction	✓	✗
WebGL	✗	✗
Request header ordering/checksum, window.navigator checksum, checksumming in general	✗	✗
Flash/Java-driven queries	✗	✗
Third-party cookies	✗	✗
JS/rendering engine differences	✗	✗
Packet inspection/clock skew (?)	✗	✗

What Chameleon and CanvasBlocker actually do is intercepting calls to properties and functions that might be related to fingerprinting. The problem here is the fact that it is not possible to determine, if accesses to those properties or functions are benign or malicious. Therefore, they can not classify a web site, saying it is tracking or it is not tracking. This is the part that this thesis tries to fulfill by mixing in entrenched detection mechanisms.

3.3. Counter-Measures

The last Sections focused on Browser-Fingerprinting and Canvas-Fingerprinting. It has been shown that these techniques have enormous potential to create fingerprints of machines that can consequently be used as track users. This Section is about counter-measures against those two techniques, with Section 3.3.1 concentrating on Browser-Fingerprinting and Section 3.3.2 focusing on Canvas-Fingerprinting. It will be shown that the most obvious measurements to hamper fingerprinting lead to even a wider fingerprintable surface. Moreover, it will be illustrated that randomization might be a good choice to defend against fingerprinting. It will also be highlighted that Canvas-Fingerprinting is extremely hard to block right now and that the best defense actually might be to involve the user.

3.3.1. Browser-Fingerprinting

Obviously, for JavaScript-based fingerprinting, which is focused on in this thesis, there exists a quite simple and absolute secure way to block it: disable JavaScript. While this would make Browser-Fingerprinting effectively worthless, it is not a real solution to the problem. With JavaScript being deactivated, the web suddenly becomes a lot uglier. In times of Web 2.0, web sites are based on several JavaScript frameworks like jQuery[14] that offer dynamic content. If this essential part of web sites is removed, it is quite likely that they become useless. However, Section 2.3.1 showed that Browser-Fingerprinting is not something that should be ignored.

As it turns out, defending against Browser-Fingerprinting is not as easy as one might think at first. There exist a lot browser add-ons that allow the modification of several features which are also often used as part of a fingerprint, for example *user_agent* or *HTTP-Header*. One can ask the question, if using these add-ons would

[14] `https://jquery.com/`

hamper Browser-Fingerprinting. The ironic part here is that a lot of such measures actually cause the exact opposite, because these measures are features by themselves and increase the uniqueness of a fingerprint, if not a lot of people also install these add-ons [11]. Actually, the best way to defend against Browser-Fingerprinting is to have the same properties, as as many other people as possible. This is the reason for iPhones being a lot harder to fingerprint, than normal computers.

Another aspect concerns Java and Flash. Although, this thesis concentrates on JavaScript-based fingerprinting, it is essential to say that these technologies are far too powerful right now. Flash should not provide as many APIs to read out browser and system specific properties. In fact, Flash makes it incredibly easy to get a list of the installed fonts, which is one of the strongest features found in [11].

A very interesting approach is mentioned in [47]. Nikiforakis et al. [47] implemented *PriVaricator*, an extension to the privacy mode that almost every modern browser supports. PriVaricator aimed to hamper fingerprinting rely on properties of the browser environment. It is also implemented directly in the browser's source code and not realized as a browser add-on or something similar. The principle behind PriVaricator was randomization. Each access to a browser property, which is potentially relevant for Browser-Fingerprinting, was intercepted and accordingly handled. Functions that only returned integer values like `offsetHeight` or `offsetWidth` were a lot easier to manipulate, than more complex functions as `toDataURL`. Therefore, the authors identified two different groups of elements that had to be considered:

1. Strategy regarding offset: For functions or properties like `offsetHeight`, `offsetWidth`, `getBoundingClientRect`, there were three different policies defined:
   a) *Zero*: Returns always zero.
   b) *Random(0..100)*: Return random values between 0 and 100.
   c) $\pm$ *5% Noise*: Return the original value $\pm$ 5% noise.
2. Strategy regarding plugins: In order to hamper the classic plugin enumeration technique mentioned in 2.3.1, a probability $P(plug\_hide)$ had been defined. This probability was used, whenever the plugins were enumerated and was applied for each plugin.

Along with these two strategies came two additional variables: *lying threshold* θ and *lying probability*. The first one defined a maximum of allowed reading attempts. If for example $\theta = 50$, the first 50 accesses to `offsetWidth` or `offsetHeight` were totally legit. The 51th and every following attempt however would trigger the specified policy. *lying probability* signaled the probability to lie, after the threshold had been reached.

Manipulating the properties and methods in this way is not without any risk. First of all, not all properties are suited to be changed. For example, the display resolution is not only a potential feature for fingerprinters, but also an important hint for web developers that use this value to display their web page accordingly. Browsing a web page on a tablet requires a different layout than visiting it with a desktop computer and a 24“ monitor attached to it for instance. When implementing the policies, the authors kept in mind that web pages should not break.

Nikiforakis et al. [47] examined how PriVaricator performs against the fingerprinting libraries of BlueCava, Coinbase, PetPortal[15] and fingerprintjs. Overall, the results of PriVaricator were pretty astonishing. It managed to delude all tested libraries in a majority of the tested combination settings. fingerprintjs, for example, was very fragile and was only able to track the authors, when the hiding probability of plugins was either zero (which means, all plugins are showed) or 100% (no plugin is exposed). Of course, this was due to the fact that fingerprintjs is not capable of font detection. As a matter of fact, the fingerprinting library from PetPortal was the most resistant in this test.
Also the breakage concerns could be ruled out. The authors tested the Alexa Top 1.000 web sites and came to the conclusion that the measurements made by PriVaricator did not break web sites. The differences with PriVaricator compared to the original web site were minimal and therefore negligible.

Also, the authors of “FPDetect: Dusting the Web for Fingerprinters” [39] did test two possible countermeasures to Browser-Fingerprinting. At first, they examined the usage of Tor Browser [49]. The Tor Project

[15] At `http://fingerprint.per-portal.eu/`, the fingerprinting framework developed by Boda et al. [48] during their study can be visited.

aims to provide the highest level of anonymization possible. In order to be able to provide protection against surveillance, Tor deployed their own browser, which is based on Firefox but implemented a lot of modifications that should ensure a higher level of privacy. A version of Tor Browser can not be fingerprinted by browser properties, because those are either not served to the user or are fixed and thus not suitable as part of a fingerprint. However, fonts are part of the operating system, which makes them kind of independent of Tor Browser and consequently a fitting candidate for fingerprinting. The Tor Project tries to circumvent this by implementing a maximum number of fonts that can be requested by a site. In [39], a bug in this system has been found that could effectively lever out this policy, but it has been fixed since then.
Another counter-measure that has been tested by the authors is Firegloves [50], a browser add-on for Firefox that tries to hamper Browser-Fingerprinting. For this purpose, it modifies the values of given properties like the screen resolution or the height and width of HTML elements to be random when accessed. The authors showed that Firegloves is not able to prevent Browser-Fingerprinting completely. This is due to several different reasons. First of all, not every relevant function or property is overwritten. For example, while the properties `offsetHeight` and `offsetWidth` are modified and therefore not suitable to be used for font detection, this can easily be bypassed by using the `getBoundingClientRect` method instead of those properties. Moreover, Flash is sill a big threat, since browser add-ons are impotent against it. To be fair, it has to be noted that Firegloves was developed as a proof-of-concept and its development has stopped a few months ago.

3.3.2. Canvas-Fingerprinting

The first and most secure counter-measure is pretty obvious: remove the possibility to extract the data from a canvas. While this will effectively prevent Canvas-Fingerprinting, one should not forget that some areas of application, for example a web app for drawing, could not be realized without this functionality [19].

Another possibility to hamper Canvas-Fingerprinting is the browser adding random pixel noise whenever the content of the canvas is extracted. Examining this approach more closely, one comes to the decision that adding simple noise can be easily circumvented by rerendering each canvas multiple times and comparing the results. Tackling this problem can be done by increasing the noise - this would, however, result in a decrease of performance, which is not desirable concerning legit applications. Consequently, Mowery and Shacham [19] come to the conclusion that adding noise is not a reasonable defense against Canvas-Fingerprinting.

Thinking about how Canvas-Fingerprinting functions, another defensive approach comes to mind. Since Canvas-Fingerprinting works because of differences produced by hardware and software, it might be a feasible attempt to force browsers to produce the exact same output regardless of their underlying hard- and software. This would have several implications for browser vendors: Firstly, they would need to ship a list of agreed upon fonts along with a library capable of rendering text (for example Pango[16]) with their browsers. Secondly, WebGL support would need to encapsulate rendering from the graphics card and use a software rendering library like Mesa 3D[17] instead. As Mowery and Shacham [19] stated, this approach might be the correct choice when the highest level of possible privacy is not debatable. Nevertheless, not using the graphics card for rendering would lead to a significant performance loss. Because of that, this attempt is not acceptable in real-life scenarios.

Another possibility to tackle Canvas-Fingerprinting at its roots is to force the browser to request user approval before he is allowed to use the `<canvas>` API. The Tor Browser [49], known for extremely high standards regarding privacy and security, implements such behavior. But also users, who do not want to use the Tor Browser, because they don't need such high standards, can force their normal Firefox to behave like that. This can be achieved by installing the browser add-on *CanvasBlocker* [46]. The add-on provides several strategies how to handle the usage of the `<canvas>` element. One strategy is the above mentioned question for user approval, another would be to block the entire `<canvas>` element. Asking for user agreement allows legit applications to use the `<canvas>` API and all of its functionalities, but impedes Canvas-Fingerprinting.

This Chapter presented related work in the field of fingerprinting techniques. For both Canvas- and Browser-Fingerprinting, the according publications were presented. Moreover, two browser extensions that aim to

[16] `http://www.pango.org/`
[17] `http://www.mesa3d.org/`

Figure 3.2.: *CanvasBlocker*'s notification when user approval is requested before the `<canvas>` element can be used.

detect those fingerprinting techniques were illustrated. Lastly, possible counter-measures that can be implemented in order to hamper fingerprinting were discussed.
The next Chapter will now try to bring fingerprinting techniques and traditional detection mechanisms together. The Chapter's main goal is to show, how these detection mechanisms can be applied in the field of fingerprinting, in order to be able to detect, if a web site is trying to fingerprint the user, or not.

4. Design

This thesis aims to detect fingerprinting techniques with the help of detection mechanisms. Therefore, a system has to be designed, developed and ultimately implemented. This Chapter will introduce the basic design of this system. Section 4.1 will derive all relevant components of the system based on a exemplary work flow and Section 4.2 will the present the resulting architectural design. After that, the components of the architectural design - Sensors (Section 4.3), data collection (Section 4.4) and logic/evaluation (Section 4.5) - are explained in detail. Section 4.6 will focus on communication. This includes the specification of possible protocols, as well as the data format used for transportation. Section 4.7 will propose two different ways to deploy the proposed system. Moreover, this Section will elucidate possible concerns regarding security and privacy.

4.1. Deriving Components

The system developed in this thesis tries to apply traditional detection mechanisms to the field of Web-Tracking, more specifically to the area of fingerprinting techniques. Its main goal is to use those detection mechanisms to predict, if fingerprinting techniques are adapted on a web site. The detection should be performed dynamically when the web site is visited. The fundamentals concerning Web-Tracking, Browser-Fingerprinting and Canvas-Fingerprinting, as well as basics regarding detection mechanisms have already been introduced in Chapter 2.

In order to make clear, how the system should function when it's ready for deployment, Figure 4.1 shows a basic work flow.

This work flow helps to determine actuators, components and communication paths. The first actuator is, of course, the user. In this scenario, the user is the one that visits a web site. He does not know, if this web site is tracking him or not. This is why he is using the detection system proposed in this thesis. When visiting a web site, he has to wait for it to load all necessary resources which can also be located on different servers. If the web site is using tracking mechanisms, more specifically fingerprinting techniques, it has to create a fingerprint of the machine. Browser- and Canvas-Fingerprinting techniques do this by using several so called features, as already explained in Section 2.3. This is where the first component of the system comes into play, the so called *sensor*. The sensors have to be able to notice that the features relevant for the fingerprinting techniques are being accessed by the page.

After all relevant features used by the web site have been extracted, the user needs to decide, whether he wants to know, if the page is tracking him or not. If he decides that that he wants to know it, there is another component required for this to be realized: *logic/evaluation*. This component is responsible for the classification of new input. It should involve several different detection mechanisms and is used to predict, if a visited web site is considered to use tracking mechanisms - it is tracking their users - or not. There is a wide range of diverse detection mechanisms. Some of them require for example a set of training data to be trained with. Others try to classify new input without the need for such data. However, this shows that the classifier heavily interacts with another component of the system: *data collection*.

The *data collection* component is responsible for all kinds of storage. This can be various data, for example information about the classified web sites. Moreover, it is thinkable to store training data here which can be used to train classifiers. This is also demonstrated in the work flow.

To sum it up, the following components have been identified:

- Sensors to extract features
- Data collection to manage a database

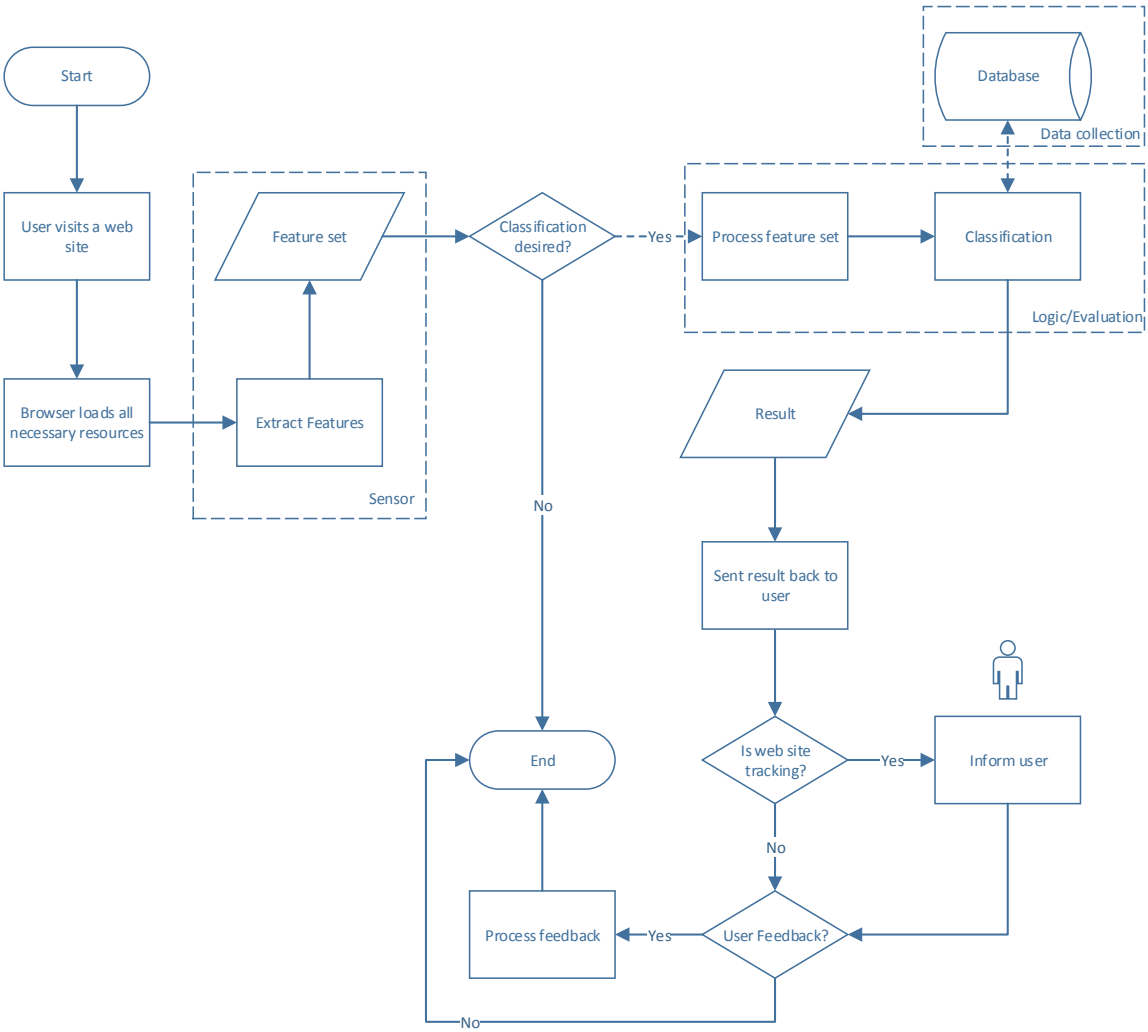

Figure 4.1.: Flowchart describing the basic way of functioning of the proposed system.

- Logic/evaluation to classify

Another aspect to consider is communication between those components. The dashed lines in Figure 4.1 show, where communication does take place.

The designed work flow from Figure 4.1 combined with the derived components along with necessary communication form a basic architectural design, which will be presented in the next Section.

4.2. Architecture

Each one of the components derived in Section 4.1 is represented as one layer in the architecture. First of all, there has to be one layer that provides information used for several purposes, including training or classifying. This layer involves the sensors. Secondly, the information provided by this layer has to be stored and managed in some way. That is the responsibility of the data collection layer. Besides, communication between these two layers is mandatory. Last but not least, one layer is responsible for evaluating the processed data. Here, evaluation is equivalent to classification. Also, communication between logic/evaluation and data collection layer has to be established.

In Figure 4.2, the fundamental design picturing these layers is demonstrated.

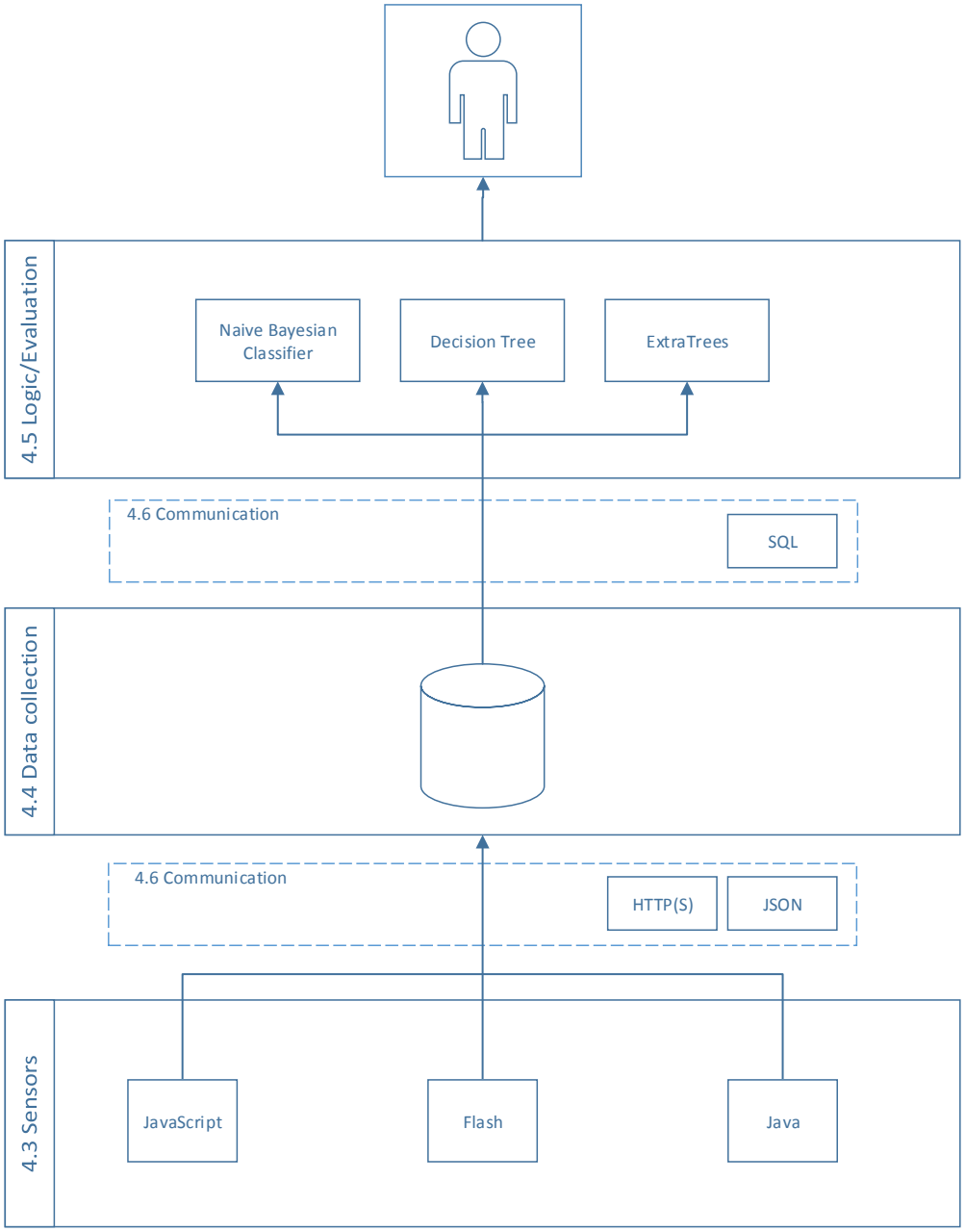

Figure 4.2.: Fundamental design of the proposed system.

As one can see, each of the introduced layers is present in the design. However, it must be understood as a fundamental idea. It can be deployed in different ways. Two of them concern the distribution of the layers. The question here is, if all layers should reside in one place, or if the system is split up and different layers are implemented in different locations.

The system always includes a client and a server, but which layers are assigned to which part is debatable and open for interpretation. The following Sections will refer to client and server as part of the communication.

4.3. Sensors

The sensors layer resides on the lowest level of the system. It holds several different sensors and provides information for the upper layers. As mentioned before, fingerprinting can be done by using different technologies, for example JavaScript, Flash or Java. For each of these technologies, one sensor has to be developed. It holds a set of relevant features and observes them. If a web site uses one of these features, the sensor must notice that and react accordingly. The reaction is completely task-reliant. In some cases, it might be enough to just log the access, in other scenarios, it might be required to take specific counter-measures, e.g. returning not the original value but a modified one.

This thesis focuses only on JavaScript-related fingerprinting. Therefore, the only crucial technology is JavaScript with Flash and Java being left out. The following part of this Section will only apply to sensors responsible for JavaScript.

Since sensors are a key-part in detection systems, there are some requirements they have to incorporate in order to be considered good ones.

1. Sensors should be “complete”.
2. Sensors should run in the background.
3. Sensors should be fast.

During this thesis, the first requirement turned out to be one of the most complex parts. Being “complete” in this context means that sensors are not allowed to miss features that were accessed on a web page. Sensors that are not guaranteed to notice the access of features, are not suitable to be used, because each feature is a small piece of the puzzle and only when all pieces are collected, detection might be possible. Therefore, is has to be ensured that sensors really are capable of identifying a web site accessing features that lie in their area of accountability. While this behavior is wished for, it became clear that this is really hard to implement. More information about this problem will be given in Section 5.1.1.
Of course, the prototype implementation developed during this work has been designed to be deployed in the wild. However, deploying this system implies several requirements. First of all, sensors have to be fast, meaning that it is absolutely crucial that they don’t delay page loading. It’s quite obvious that this would lead to the detection system being ignored by almost every user. Delaying page loads is not an acceptable trade off. The second consequence is the fact that sensors have to run in the background, performing their task without the user noticing them. Again, this requirement directly results from the objective that using the detection system should be as easy as possible.

4.3.1. Defining Suitable Features

Ideally, for each possible technology, one sensor is drafted and finally implemented. The sensor provides features of its observed technology and forwards them to the data collection layer. As already mentioned, only JavaScript-based features will be taken into account.

The selection of suitable JavaScript-related features is based on the *fingerprintjs* library and the Chameleon browser extension. Moreover, the features used in the initial project concerning Browser-Fingerprinting - “Panopticlick” - are also included. All three of them are already illustrated in Section 2.3.1.

The features used in all those projects can be divided into two different types:

- *MemberExpressions*: Those features represent accesses on member variables of objects. A classic example for this would be `navigator.plugins`. They consist of two different parts: the *object* (in this case: navigator) and the *property* (here: plugins).
- *FunctionCalls*: These are quite similar to MemberExpressions, however their property is not a variable of the object, but a function that is called on the object. A typical representative is `toDataURL`, a function that is called on the `context` object, gained from the `<canvas>` element.

As one can see, each feature consist of a tuple `(object, property)`. Table 4.1 shows all considered features along with their objects and properties.

Table 4.1.: Relevant features for this thesis.

Type	Object	Property
FunctionCall	HTMLCanvasElement	toDataURL
FunctionCall	HTMLCanvasElement	getContext
FunctionCall	CanvasRenderingContext2D	fillText
FunctionCall	CanvasRenderingContext2D	strokeText
FunctionCall	CanvasRenderingContext2D	getImageData
FunctionCall	WebGLRenderingContext	getParameter
FunctionCall	WebGLRenderingContext	getSupportedExtensions
FunctionCall	Date	getTimezoneOffset
MemberExpression	navigator	plugins
MemberExpression	navigator	userAgent
MemberExpression	navigator	language
MemberExpression	navigator	cpuClass
MemberExpression	navigator	platform
MemberExpression	navigator	doNotTrack
MemberExpression	screen	height
MemberExpression	screen	width
MemberExpression	screen	colorDepth
MemberExpression	window	sessionStorage
MemberExpression	window	openDatabase
MemberExpression	window	localStorage
MemberExpression	window	ActiveXObject
MemberExpression	window	indexedDB
MemberExpression	window	devicePixelRatio
MemberExpression	window	innerWidth
MemberExpression	window	innerHeight
MemberExpression	HTMLElement	offsetHeight
MemberExpression	HTMLElement	offsetWidth

This sample of features tries to match to most popular ones used in recent studies. Both, Browser- and Canvas-Fingerprinting related features are included in the feature set.

Canvas-Fingerprinting is a special form of Browser-Fingerprinting and it therefore requires some special attention. When examining Canvas-Fingerprinting in great detail, it becomes clear that applying it is usually done in a particular way.

1. Create a `<canvas>` element: `document.createElement('canvas')`
2. Get the `context` element used for drawing: `getContext`
3. Draw something on the canvas: `fillText`, `strokeText`
4. Extract the pixels produced: `toDataURL`, `getImageData`

However, the first step is kind of optional, depending on whether there is already a `<canvas>` element present in the DOM[1]. As the above mentioned Enumeration shows, almost every FunctionCall present in the feature set is relevant for Canvas-Fingerprinting. Obviously, some combinations of FunctionCalls, for example `toDataURL` alongside with `fillText` seem to be a strong indication for Canvas-Fingerprinting. The methods `getSupportedExtensions` and `getParameter` are often used in WebGL-related fingerprinting attempts.
Another technique yielding high entropy, is the list of installed fonts, which is accessible via font enumeration. By logging the `offsetHeight` and `offsetWidth` methods, it might be possible to draw conclusions whether this technique is utilized or not. This requires to count how often those methods are executed. Font enumeration heavily relies on the `offsetHeight` and `offsetWidth` functions of `<span>` elements. `<span>` elements are a sub-type of HTMLElements, so whenever those functions are called, the counter is incremented. Very high values for those counters might be a hint for font enumeration being used on the web site.

4.4. Data Collection

The data collection layer is the middle ware of the system. Its objective is to store all relevant data of the system. This involves a wide variety of information sources, like features of web sites or data applicable for reporting purposes.

4.4.1. Database System

Whenever storing data is the topic, the first thought that comes to mind are databases. They provide huge advantages compared to other storing technologies like simply writing files containing the data. Especially when the amount of data stored is increasing over time, databases can play out their strengths concerning speed and data management.

The striking advantages of databases are the reason for the data collection layer to use a database as well. There are several different database management systems in the wild and they can be divided into two classes: *relational* and *non-relational* (NoSQL, also interpreted as "Not only SQL") *databases*. They differ in the way they store the data, which results in some operations being faster in relational databases and some being faster in non-relational databases. Relational databases, for example, use tables for data storage.

NoSQL databases have the following advantages over relational databases [51]:

1. Some providers (Riak, Cassandra) are able to handle hardware failures
2. Scalable
3. Wide range of data models
4. Faster, more efficient and flexible
5. No database admins required
6. Very high speed in terms of progression

The disadvantages over relational databases are stated as follows:

1. Still in development
2. No default query language (like SQL)
3. Not all are ACID[2] conformable
4. No default interface
5. Service is difficult

[1] Document Object Model
[2] Atomicity, Consistency, Isolation, Durability

In general, NoSQL databases provide huge upsides concerning speed and scalability. However, being a rather new technology, they lack a standard query language like SQL. Relational databases are approved and widely used in a lot of projects. With SQL, they provide a standard query language being well-known by a broad majority of technicians. Relational databases should be used, when the data to be stored can easily be mapped on tables. As the next Section will show, this is the case in this thesis. NoSQL databases are generally speaking the better fit, when the data is unstructured.

One of the most important advantages of NoSQL over relational databases is scalability. They scale by distributing them on numerous servers. Relational databases provide difficulties with such scaling. Their performance is improved by updating the server's hardware they are running on [52]. To sum it up, this thesis will use a relational database. The reasons for this are the presence of a query language like SQL and the fact that the data to be stored is structured. Moreover, it won't be necessary to scale the database extraordinarily, because the number of features will stay manageable and won't overexert the relational database.

4.4.2. Database Content

As already mentioned, one purpose of the data collection layer is storing web sites and their according features, which play an essential role when it comes to classifying.

If one visits a web site, the sensors are responsible for the extraction of features of their area of application. This means that each sensor delivers a set of features for one web site. It is enough to determine, if a given feature is used by the web site or not; the number of its occurrences is not considered at the moment. Moreover, each feature can occur on more than one web site. This results in many-to-many relationships between web sites and features. Moreover, each feature has a specific type. The basic relational database design for the system presented in this Chapter is illustrated in Figure 4.3.

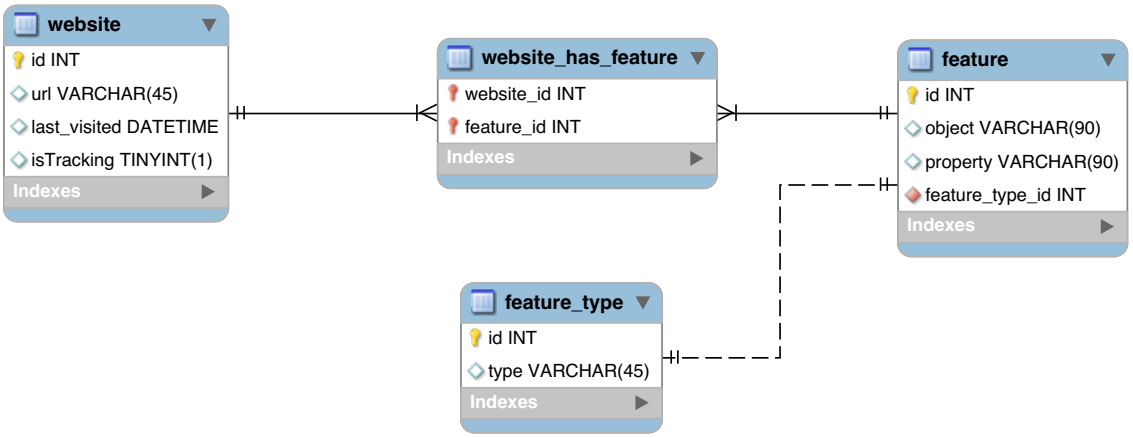

Figure 4.3.: Database scheme for the data collection layer.

The *web site* table is used to store a web site, which consist of an *id*, the *url*, the date and time when it has been parsed and a flag, which defines if this web site is considered to track users or not. The *feature* table stores all features found on the different web sites. Each feature contains an object and a property. Please note that this is only valid for JavaScript features. Each web site has numerous features and each feature can occur on several web sites; this results in a many-to-many relationship between *website* and *feature*, which is modeled by introducing the *web site_has_feature* table. The *feature_type* table holds different types of features. JavaScript-based features for example are MemberExpressions and FunctionCalls. Using this table ensures that the database can be easily extended and new feature types can be added. Each feature has exactly one feature type.

Section 4.6 will state that the format of transferred data is JSON. To bring JSON and the database scheme proposed in this Section together, Listing 4.1 presents an example of the features found on the web site `http:`

`//www.pof.de/`. As one can see, FunctionCalls and MemberExpressions are represented using a JavaScript object. Their keys represent the `object` part of a feature. Each feature forms a JavaScript object itself with the key being the `property` part of it. The value of the objects denotes the total number of uses. This additional information is not taken into account right now, but might be of use in the future. At the moment, the only interesting aspect is, whether a feature has been used or not.

Listing 4.1: Exemplary data in the JSON format.

```
{
  "url": "http://www.pof.de/",
  "functionCalls": {
    "Date": {
      "getTimezoneOffset": 4
    },
    "HTMLCanvasElement": {
      "getContext": 1,
      "toDataURL": 1
    },
    "CanvasRenderingContext2D": {
      "fillText": 2
    }
  },
  "memberExpressions": {
    "navigator": {
      "plugins": 19,
      "userAgent": 4,
      "platform": 1,
      "language": 1
    },
    "window": {
      "ActiveXObject": 2,
      "innerWidth": 1,
      "innerHeight": 1
    },
    "screen": {
      "width": 3,
      "height": 3,
      "colorDepth": 3
    },
    "HTMLElement": {
      "offsetHeight": 1,
      "offsetWidth": 1
    }
  }
}
```

4.5. Logic/Evaluation

The logic/evaluation layer is the place, where detection mechanisms are implemented. It can host several different detection mechanisms. Some of them might require training before they are capable of classifying new input. Moreover, it exposes an interface that expects a web site and their features as input. This input is then classified using one of the detection mechanisms. The result of this process represents the classifier's estimation of the input's class - tracking or not-tracking.

The idea behind this classification process is the following: There exist two different types of web sites: the ones that are tracking and the ones that are not tracking their users. Each of these web sites does use a specific set of features. The basic assumption of this thesis is that the tracking web sites and not-tracking web sites differ in three areas:

1. Different features are present in the tracking web sites and not- tracking web sites.
2. Sites within the same category tend to use the same features.
3. Sites that are considered to track, are believed to use less features.

Classification of web sites is performed based on these assumptions.

4.6. Communication

Communication between connected systems is an incredibly complex topic. Therefore, the OSI[3] reference model (will be called OSI model from now on) has been introduced. This model provides the basic architecture for communication between systems. It consists of seven layers, which are depicted in Table 4.2 [53, 54].

Table 4.2.: OSI Reference Model for communication.

Layer name	Protocol(s)
Application Layer	HTTP, SMTP, DNS
Presentation Layer	
Session Layer	
Transport Layer	TCP, UDP
Network Layer	IP, ICMP
Data Link Layer	DSL, Ethernet
Physical Layer	

Choosing the protocol for communication depends on which layer the communication should be implemented on. For example, the communication can be realized using the connection oriented TCP[4] protocol. Another option is UDP[5], a connectionless protocol, which basically offers the same functionalities like TCP, but with slight advantages regarding latency, but with disadvantages in terms of reliability. Both of those protocols are located on the Transport Layer.
Also, using HTTP[6] is a reasonable possibility. HTTP is the basis for the Internet and is located on the Application Layer of the OSI model. It is used to communicate with web servers using methods like *GET*, *POST*, *PUT*, *DELETE* and some others [53] that provide different functionalities.
HTTP seems to be the ideal choice for the system presented in this thesis, because implementing web servers is rather easy and communicating with them is a standard task that can be achieved with a wide variety of libraries. Moreover, security can be achieved using HTTPS[7], as indicated in Section 4.7.2.

Since HTTP is the protocol of choice, the usual way to transmit data using this protocol is the *POST* method, whose body is used for data transmission. The *POST* method does not impose requirements on the data format. In the context of the system, the exchanged data will mostly consist of features. As discussed in Section 4.3.1, the feature type relevant for this thesis consist of a tuple `(object, property)`, which can be interpreted as key-value pairs. For this kind of data, two data formats are suitable: *XML*[8] and *JSON*[9]. XML is a really powerful markup language and it can be used to encode any file of any format. It is even possible, to define completely new types. These possibilities do come with downsides: XML is relatively complicated to read and does involve a lot of boilerplate code, which ultimately leads to more bytes being transformed when using XML. The biggest competitor to XML right now is JSON. This data format has a much simpler syntax and less overhead. Nevertheless, it is not possible to define own data types, but only standard types like strings or integers are allowed. Being overall smaller automatically means that less bytes are transported. Moreover, JSON is considered to be better in terms of performance. One of its biggest upsides however is the fact that it's build in JavaScript, which makes using it straightforward. Since the data types being transferred in this system are only primitive ones, the biggest advantage of XML over JSON - the ability to define own data types - is not utilized. Moreover, being very lightweight is also a convenient aspect that has to be taken into account. These points led to JSON being used as data format for transmitted data.

[3] Open Systems Interconnection
[4] Transmission Control Protocol
[5] User Datagram Protocol
[6] Hypertext Transfer Protocol
[7] HTTP over SSL/TLS
[8] Extensible Markup Language - `http://www.w3.org/XML/`
[9] JavaScript Object Notation - `http://json.org/`

4.7. Discussion

The last Sections introduced the basic design of the proposed system and its components were presented. The architecture of the system must be critically discussed. Two different variants to deploy the system, the monolithic and the distributed approach, are presented in Section 4.7.1. Moreover, Section 4.7.2 will examine the architecture with a special focus on security and privacy. Lastly, the security and privacy implications of the deployments models will be discussed.

4.7.1. Deployment Variants

Monolithic Approach

One approach to structure the system is a monolithic approach. This means that each user of the detection system hosts a full-fledged version of the whole system. There is no data shared between the users of the system. Figure 4.4 shows the basic idea of such a structure. As one can see, every single layer of the system - sensors, data collection and logic/evaluation - is located at the same place and forms one unit.

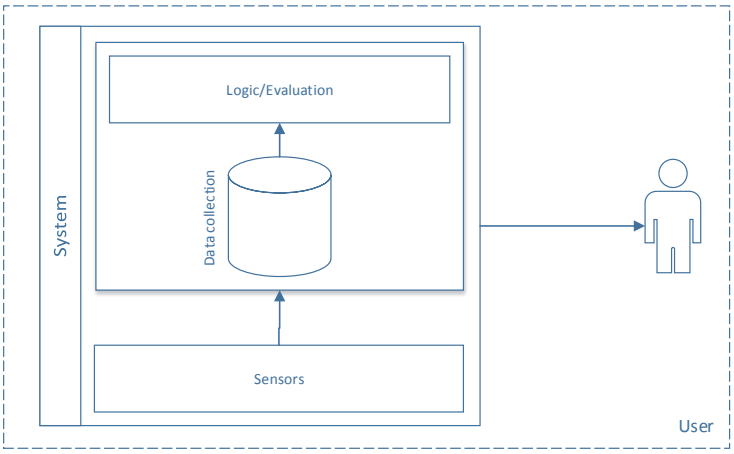

Figure 4.4.: Basic idea of a monolithic system.

The advantages of such a system are as follows:

1. Updates to the system can be easily achieved, because the whole system is deployed as one unit.
2. Communication becomes less complicated, because the communication partners are part of the same unit. Therefore, it is not necessary to transport data via the Internet or some other network. This also eases the requirements concerning security.
3. The user has every part of the system under control.

However, a monolithic system like that also comes with several disadvantages. They are depicted below.

1. Since the whole system is deployed as a unit, all information required by the system has to be shipped with it, involving data collection, logic/evaluation layers, as well as sensors. Depending on what those layers contain, the disk space required could potentially be very high.
2. As already indicated, it is thinkable to consider user feedback when classifying new input. If this user feedback is stored for future use, those changes are only locally. Other users do not profit from them, because every user has its own copy of the system and local changes are not distributed to other users.

Distributed Approach

The distributed approach is an alternative to the monolithic one. In this approach, the sensors are located on different places, like the user's machine or at completely different locations. The difference to the monolithic approach is that the distributed part of the system only consists of sensors. The other layers - data collection and logic/evaluation - are located at a single place, usually a server. However, the operator of this server is not specified. This can either mean that the server is self-hosted or that it is delegated to another service provider. The basic idea of this approach is that the sensor logs the relevant features and sends them to a central server. The server receives this set of features and classifies them using the logic/evaluation layer. The result of this process is then sent back to the user. Figure 4.5 illustrates this approach. As it becomes clear in this image, there is only one central server holding the data collection and logic/evaluation layers. Please note that such a system also has a central component: the server hosting data collection and logic/evaluation layers.

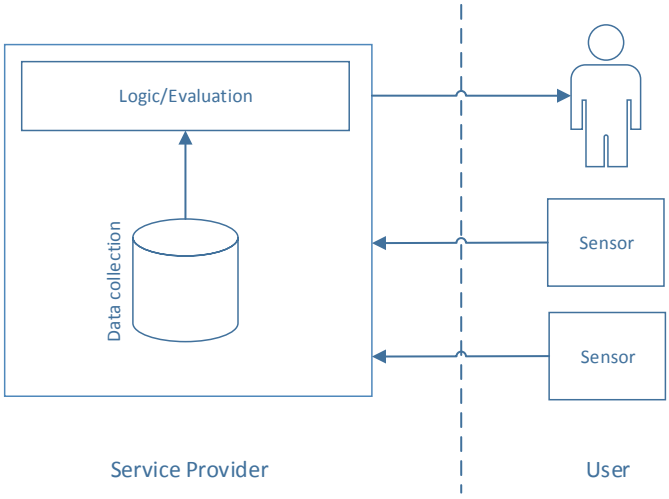

Figure 4.5.: Basic idea of a distributed system.

Such a distributed system does have several upsides, which are presented below:

1. Clients only consist of sensors and are therefore relatively small and easy to implement.
2. If user feedback is taken into account (meaning that the user can give feedback on the classification), it can be collected at a single point. This implies that classified input can be combined with the user feedback and stored for possible future use. Since there only exists one data collection and logic/evaluation layer, changes here are reflected directly to every user of the system.

Most disadvantages of the distributed approach concern its central component: the server. Since it is a vital point of the system, it becomes the single point of failure. If the server crashes, the whole system collapses and classification is no longer possible. Moreover, the server is an interesting suspect for attackers, since all the system's knowledge is combined here. Overall, it is safe to say that the server needs special treatment in distributed system. There are several requirements concerning privacy, security and availability.

4.7.2. Security and Privacy

Security and privacy both play an important role and they have to be taken into account when designing such a system. This Section will focus on deliberations and requirements concerning security of the architecture depicted in Figure 4.2. Each one of the layers will be examined in regards of common protective goals. Moreover, concerns regarding privacy will be discussed. It will be shown that it is absolutely necessary to focus on privacy, because otherwise, the detection system is in fact a better tool for Web-Tracking than most trackers are using right now.

Security

Almost every system involves data and information that is worth protecting. Nevertheless, protecting this data is mostly a very complex task with even the smallest mistakes having fundamental repercussions, like data leaks of sensitive and personal information. In order to provide a guideline on how to achieve security for sensitive data, several protective goals have been defined [55]:

1. *Authenticity*
2. *Integrity*
3. *Confidentiality*
4. *Availability*
5. *Non repudiation*
6. *Accountability*
7. *Anonymization*

Confidentiality, Integrity and Availability form the so called *CIA triad* [56]. Due to their importance in information security, they are going to be explained specifically.

- *Confidentiality*
  Only authorized users may have access to data and information. This has to be ensured for both stored data and also during its transport. For the purpose of ensuring confidentiality, it is necessary to define and establish access levels for information. Consequently, it is specified, which users have access to which information.
- *Integrity*
  Data integrity is ensured, if unauthorized users are not able to alter or delete data undiscovered. Moreover, it has to be ensured that even if data have has changed, those changes can be undone.
- *Availability*
  Availability means that the information must be available when it's needed. Therefore, every authorized user must be able to access the information he needs. This principle also counts for authentication mechanisms used to protect the information and access channels utilized to access it.

When designing such a system, all of the above mentioned principles have to be taken into account accordingly. Examining the proposed architecture from Figure 4.2 leads to several potential weak spots.
All layers respectively components of the detection system are potential gateways for attackers and therefore, every component needs special attention in terms of security. But not only the layers are interesting targets for attackers, but also the communication between them. Communication needs to be secured respectively. It is always one of the most interesting weak spots. Each of those potential points of application - sensors, data collection, logic/evaluation and communication - need to be secured using the CIA triad. The following Paragraphs will discuss confidentiality, integrity and availability for each of them.

Communication Securing communication is one of the biggest topics in information security. Several aspects have to be considered. Luckily, this topic is researched extensively and there exist several reliable techniques to ensure confidentiality and integrity [55]. In order to detect, if data has been modified, cryptographic hash functions are usually utilized, for example *HMAC*. For the purpose of ensuring confidentiality, the mechanisms of choice are encryption techniques. One possible scenario in this context that really requires data integrity is a man-in-the-middle attack. This type of attack enables a possible attacker to manipulate the data on its way to the server. This is an extremely critical point, since the data is used as input on the server. An attacker might be able to alter the request content in order to trigger server-side behavior that is not intended. A classic example for this attack would be SQL injection. The approved way to circumvent this attack vector is to encrypt the communication, for example by using TLS[10] [53]. This leads to the attacker being unable to manipulate it.

[10]Transport Layer Security

Data collection and logic/evaluation Since the data collection layer is responsible for storing all relevant data of the system, data integrity is obviously a big topic. It has to be guaranteed that the information can not be modified or deleted by unauthorized users. Altering the data here can lead to the logic/evaluation layer misclassifying new input, which would result in the system to be useless.
Moreover, the principle of confidentiality has to be ensured. This means that only authorized users should be allowed to use the detection system. The architecture proposes a client-server model and therefore, the server has to provide several APIs that can be called by the client, for example to trigger classification. The emerging problem here is the fact that there has to be a mechanism allowing the system to identify legitimate requests. If there is no such authentication system, everybody could send requests to this API and thus overextend the server's capabilities causing a crash of the server.
One way to implement such an authentication system are *security tokens*. This principle is often used in mobile applications and Google offers a similar service using the OAuth 2.0 protocol [57]. The concept behind security tokens is simple. Each user that wants to use the API has to request a security token at first. This can be achieved in various ways, e.g. by registering at a web portal. From now on, each request contains not only the relevant data but also the security token. The server knows about all legitimate security tokens and only accepts requests with one of them attached to it. This mechanism ensures that only requests from clients with a valid security token are processed and consequently, confidentiality is ensured. Figure 4.6 shows the basic idea of security tokens.

Figure 4.6.: Basic concept of security tokens.

Of course, the basic concept of security tokens shown in Figure 4.6 is also a potential weak spot and has to be secured accordingly. Again, it is necessary to guarantee safe communication. Also, the other parts of the system - for example the server generating the security tokens - needs special treatment in terms of security. The details of this process are not included in this thesis and are therefore not discussed.
The remaining principle is availability. Since the logic/evaluation and data collection are key-parts of the system, availability needs to be guaranteed. If only one of those parts fails, the whole system becomes useless. A possible mechanism would be replication.

Sensors Availability of the sensors is required for the system to work properly. Depending on the numbers of sensors, it might be able to compensate the failure of a specific number of sensors. Nevertheless, the accuracy is always suffering when not all sensors provide their information for the upper layers. One way to achieve this is redundancy, meaning that every sensor is utilized multiple times on different locations.
The next part of the CIA triad concerns confidentiality. The importance of this principle here is debatable. On the one hand, it might be of interest, which sensor provided what information. On the other hand, this additional information might be of no use and is therefore not needed.
The last principle is integrity, which is very important for sensors. As has already been mentioned, the sensor provides information, which is used for the classification. It's quite obvious that the data integrity of this information has to be ensured, because if the information is changed, the classification would be performed based on wrong data and would therefore be of no use.
Security of sensors strongly depends on their location. If they are under control of the system's user, the CIA triad is way easier to establish. If they are located on a remote device, for example a router in the network

or another, central gateway, confidentiality, integrity and availability suddenly become a lot more difficult to maintain.

Privacy

Privacy also plays an important role in this context, especially since this thesis tries to detect Web-Tracking, a technique that is often criticized for violating privacy.

Section 2.1 illustrated the enormous value of Web-Tracking for trackers. It's well known that the industry is worth several billion dollars and because of that, trackers make huge efforts to improve their tracking mechanisms - for example by implementing new fingerprinting techniques that were already discussed during this thesis. Those techniques provide additional jeopardy, because they are incredibly hard to block and the user does not notice them at all. One of the central aspects in the Web-Tracking practice is the collection of the list of visited web sites of a user. This list contains a huge amount of sensitive data, which enables them to extract information concerning health status, political attitude, consumer behavior and much more. As the thesis has shown so far, trackers are absolutely keen on learning as much information about a user as possible.

The proposed system offers several points of criticism in regard to privacy. One of the salient points in this area is deeply anchored within the basic way of functionality of the detection system. The system uses sensors to extract relevant features of the web site and forwards them to the data collection and logic/evaluation layers in order to get back a classification of the web site. This design has some weak points concerning privacy. First of all, the data collection and logic/evaluation layers get to know the URL of the web site. If the system is now able to recognize users, for example by using the IP address of the request or by utilizing the security token, the system is in fact capable of building a list of visited web sites for a given user and eventually creating a user profile. Obviously, this is exactly the same as current web trackers try to achieve. As a result, the necessity to transmit the URL alongside its features has to be reconsidered. There exist several possibilities to mitigate this problem:

1. Don't send the URL alongside its features.
   Apparently, this would circumvent the problem as a whole. However, this would also lead to the inability of the system to evaluate possible user feedback.
2. Break linkability of users and web sites.
   This is one of the most important things being absolutely necessary to implement. It must be ensured that users of the system can not be linked to their visited web sites. Therefore, every mechanism enabling recognizing users must be deactivated (no IP address, no linkability between security tokens and users).
3. Anonymize transmitted data.
   The idea behind this approach is to anonymize all data being transmitted to the evaluation part of the system. One possible approach to achieve this goal is to transport the URL only as a hashed value. By doing so, it is not possible to draw conclusions from the hashed value to the original URL of the web site. Of course, the second point mentioned must still be implemented. One advantage of this approach is that user feedback could nevertheless be taken into account.

As became clear, privacy is a big concern in this system. If it is deployed in the wild, there are several aspects that need to be implemented properly to ensure the privacy of the system's users.

Influences of the Deployment Model

Section 4.7.1 presented two different variants to deploy the system: the monolithic approach and the distributed approach. Of course, the choice of the model directly influences security and privacy concerns.

Monolithic Approach In terms of security and privacy, the monolithic approach is less critical. This is because of the user having all parts of the system under his control. The first difference here concerns communication. Since all layers reside on the same system, communication can be realized without the need of a another network like the Internet. This has severe implications on the security techniques required. Of

course, integrity remains very important. But since the user is the only one having access to the system, this part is not as hard to preserve as it would be, if the system can be accessed from the outside. The same is true for confidentiality. Availability is nonetheless required for the system to work as designed. But again, since the user is under control of every part of the system, it is his responsibility to ensure the availability of all parts.

The monolithic approach is also favorable from a privacy point of view. All required parts are located on the same system and no data is exposed. Therefore, it is not possible to access the data collected by the system from outside and consequently, nobody is able to create a user profile. Of course, all the sensitive data is sill collected, but since only the user can access it, it are not as endangered as they would be in a distributed approach.

Distributed Approach The distributed approach imposes several requirements concerning security and privacy. The main reason for this is the fact that not all data is located on the same system, but a crucial part - the server involving data collection and logic/evaluation - is outsourced to a remote server. This means that the user does not have full control over the system, but in fact only over the sensors, if that. In the distributed approach, all principles presented in Section 4.7.2 are extremely important. Communication needs to be ensured using appropriate techniques in order to be able to guarantee confidentiality and integrity. Moreover, securing the server is inevitable. Proper mechanisms to enforce confidentiality and integrity on the server have to be implemented. This involves an eligible authentication process (e.g. security tokens). Another important aspect is availability. Since all users of the detection system now utilize the same server for classification, ensuring availability is even more important.

Nevertheless, even bigger implications concern privacy. In the monolithic approach, the data collected by the detection system does not leave the user's machine. In the distributed approach, this is different. Classification requires the detection system to send the collected data to a remote server. As mentioned in Section 4.7.2, those requests contain incredibly sensitive data that is a perfect fit for the creation of user profiles, including a list of visited web sites. In fact, this information is exactly the type of data, every provider of Web-Tracking is absolutely keen on collecting. Therefore, appropriate mechanisms have to be implemented. Most importantly, the linkability between users and the visited web sites has to be disrupted. Additional challenges arise, if the remote server belongs to a third-party supplier, because this would mean that all the sensitive data collected by the system is not under control anymore.

5. Implementation and Evaluation

5.1. Implementation

This Section will focus on the implementation of the system proposed in Chapter 4. It needs to be highlighted that the implemented prototype utilizes the distributed approach. This means that there exist several sensors that communicate with one central server, which holds the data collection and logic/evaluation layers.
This Section will contain in-depth knowledge about the implementation of sensors (Section 5.1.1), data collection (Section 5.1.2) and evaluation (Section 5.1.3). However, not the entire source code will be printed here. Instead, only basic ideas of the implementation are provided. The interested reader is advised to further explore the source code of the prototype implementation.

5.1.1. Sensors

Sensors can be implemented on different levels. For example, Mowery and Shacham [19] opted to add their sensor functionality directly in the browser's source code. This has several advantages: Firstly, it is guaranteed that the sensor is "complete". Modifying the browser's source code ensures that the modifications can not be altered by any means, especially not by JavaScript code shipped with web sites. Secondly, implementing the sensor on this level also facilitates the realization of counter-measures, as has already been shown in Section 3.3.1. However, it also comes with a severe disadvantage: changing the source code of the browser results in a completely different browser version that has to be distributed separately. Moreover, the browser would be in competition with approved browsers. Also, the new browser must be constantly maintained, which requires huge resources regarding money and manpower. Last but not least, the users would have to be convinced to take the new browser and not the one they are used to. This is an incredibly difficult task, because humans are creatures of habit and tend to not change things that work right now.
Nevertheless, there also would be a possibility to avoid the necessity to deploy an entirely new browser. If one would be able to to persuade the big browser vendors like Microsoft, Mozilla and Google that they should implement these changes in their browsers, a huge user base could be affected directly. One possibility is for example to implement the modifications in a way that they are only applied when the private mode of the browser is activated. To be fair, one has to highlight that browser vendors are constantly seeking to decrease the fingerprintable surface of their browsers, but some things can simply not be altered due to compatibility reasons.

Another possibility to implement sensors are browser extensions. As already mentioned, they have only access to JavaScript and not to other technologies like Flash or Java, which is not a problem for this work, since those technologies are not relevant. Moreover, they also come with a big downside. If a tracker is putting great effort in his tracking mechanisms, the browser extension's efforts to detect and maybe block Web-Tracking can be circumvented. More on this topic will be discussed in Section 5.1.1. Nevertheless, browser extensions also provide a huge advantage: usability. They are rather easy to implement and - even more importantly - incredibly simple to install and deploy. Users don't need to change their browser, they can just add the new browser extension. This is an upside that should not be underestimated. Balancing pros and cons of both ways to implement sensors, the decision was made to develop a browser extension that serves as the JavaScript sensor in the detection system.

Despite the way they are implemented, all sensors in the distributed architecture do have in common that they should send the collected features to the server, where this web site and its features are processed accordingly. The server therefore provides different interfaces, like one that classifiers the input and one that adds the input to the training data.

This Section will show two different ways to log features used on a web site. Both will be explained in detail and their advantages and disadvantages will be depicted.

Static Source Code Analysis

The first possibility is to do a static source code analysis of the JavaScript code embedded in the web site. The idea is to built the Abstract Syntax Tree for each available code snippet. The Abstract Syntax Tree (AST) is a representation of the source code in form of a tree. Each node denotes a special construct present in the source code. For example, two of those constructs are called *CallExpression* and *MemberExpression*. The full AST specification of JavaScript can be found on [58]. The AST representation for the source code in Listing 5.1 is displayed in Figure 5.1[1].

Listing 5.1: Exemplary source code used for AST representation.

```
1 var txt = 'http://www.plentyoffish.com';
2 ctx.textBaseline = "top";
3 var data = canvas.toDataURL()
```

This example is a great way to demonstrate the hierarchic structure of ASTs. The first line of code can be found in the leftmost branch. The black node labeled with *VariableDeclaration* shows the type of this statement. As the name suggests, it is variable declaration, with the name of the variable being *txt* and its content being a string *http://www.plentyofish.com*.
The second line of code shows an ExpressionStatement, which is represented as the middle branch of the AST. The interesting part here is the MemberExpression. From the AST point of view, `ctx.textBaseline` is exactly the same as for example `navigator.plugins` or numerous other features defined in Table 4.1.
The last line of code in this example shows a FunctionCall. The interesting part of the rightmost branch depicting this code line is the node labeled with *CallExpression*. This kind of node represents FunctionCalls. In the AST representation, a FunctionCall is a combination of a MemberExpression (the left branch) and arguments (the right branch). Since this FunctionCall does not have any arguments, the attached arguments node does not provide any child nodes for specific arguments.

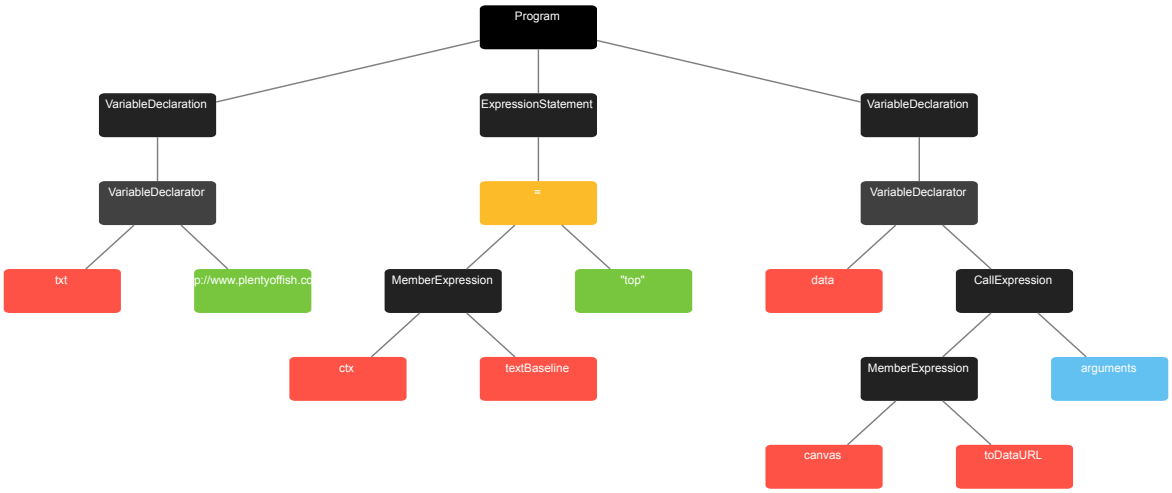

Figure 5.1.: AST representation of JavaScript code.

Transforming JavaScript code into an tree-like data structure comes with the advantage of being able to iterate through it and handle MemberExpressions and FunctionCalls accordingly. Transformation from code into an AST representation is realized by several libraries, most notably *Esprima*[2] and *Acorn*[3]. Both libraries convert

[1] The AST has been built with `http://jointjs.com/demos/javascript-ast`
[2] `http://http://esprima.org/`
[3] `https://github.com/marijnh/acorn`

the code into an AST that is built of JavaScript objects. Parsing the JavaScript code presented in Listing 5.1 with Esprima results in the AST representation shown in Listing 5.2.

Listing 5.2: AST representation built by Esprima.

```
{
    "type": "Program",
    "body": [
        {
            "type": "VariableDeclaration",
            "declarations": [
                {
                    "type": "VariableDeclarator",
                    "id": {
                        "type": "Identifier",
                        "name": "txt"
                    },
                    "init": {
                        "type": "Literal",
                        "value": "http://www.plentyoffish.com",
                        "raw": "'http://www.plentyoffish.com'"
                    }
                }
            ],
            "kind": "var"
        },
        {
            "type": "ExpressionStatement",
            "expression": {
                "type": "AssignmentExpression",
                "operator": "=",
                "left": {
                    "type": "MemberExpression",
                    "computed": false,
                    "object": {
                        "type": "Identifier",
                        "name": "ctx"
                    },
                    "property": {
                        "type": "Identifier",
                        "name": "textBaseline"
                    }
                },
                "right": {
                    "type": "Literal",
                    "value": "top",
                    "raw": "\"top\""
                }
            }
        },
        {
            "type": "VariableDeclaration",
            "declarations": [
                {
                    "type": "VariableDeclarator",
                    "id": {
                        "type": "Identifier",
                        "name": "data"
                    },
                    "init": {
                        "type": "CallExpression",
                        "callee": {
                            "type": "MemberExpression",
                            "computed": false,
                            "object": {
                                "type": "Identifier",
                                "name": "canvas"
                            },
                            "property": {
                                "type": "Identifier",
```

```
                                "name": "toDataURL"
                            }
                        },
                        "arguments": []
                    }
                }
            ],
            "kind": "var"
        }
    ]
}
```

Esprima and Acorn both create the same output, which can be traversed from top to bottom. A library that supports the user in this process, is *Estraverse* (`https://github.com/estools/estraverse`). It provides two fundamental functions [59]:

- `enter`: Called, when a node is entered.
- `leave`: Called, when a node is left.

Both functions have the same signature: `function(node, parent)`, where `node` is the current examined node and `parent` denotes its parent node.

Using these libraries enables the extraction of every relevant part of the AST. In this thesis, only two kinds are important: FunctionCalls and MemberExpressions. To get those, one has to built the AST of a code snippet using Acorn or Esprima and traversing the resulting AST with estraverse. Listing 5.3 shows, how FunctionCalls and MemberExpressions can be extracted with this combination of libraries.

Listing 5.3: Parsing an AST with estraverse.

```
1  estraverse.traverse(ast, {
2      enter: function (node, parent) {
3          if (node.type == 'MemberExpression') {
4              var object = node.object.name;
5              var property = node.property.name;
6          }
7          else if (node.type == 'CallExpression' &&
8              node.callee.type == 'MemberExpression') {
9              var object = undefined;
10             if (node.callee.object != 'undefined') {
11               object = node.callee.object.name;
12             }
13             var property = node.callee.property.name;
14     }
15 });
```

Please note that the *if-clause* in line 10 is necessary, because FunctionCalls do not necessarily require an object to be called on. Moreover, it would be possible to handle arguments of the FunctionCall accordingly. This part is left out in this example. Nevertheless, parsing arguments is not a simple task because each argument can be a MemberExpression, FunctionCall, ... on its own.

Transforming JavaScript code into its AST and using this to extract features does have several advantages:

1. Extracting features is very simple, as shown in 5.3. In fact, this code is enough to extract MemberExpressions and FunctionCalls. Note that this also includes the ones not relevant for tracking purposes.
2. Static analysis of source code is a very powerful mechanism with numerous additional possibilities, like searching for malicious code or looking for particular code patterns.
3. Arguments can also be taken into account. This might be of interest in some cases, for example the `getContext` method is usually called with one parameter "2d" when used in Canvas-Fingeprinting.

However, static analysis also comes with some disadvantages, especially when used in the context of detecting Web-Tracking.

1. It is difficult to catch the entire JavaScript code embedded in web sites. The reason for this is that a lot of third-party libraries are downloading even more JavaScript files dynamically in the background. In order to be able to parse these files, one has to intercept outgoing requests. While this is possible in browser extensions, it's not very easy to achieve. Moreover, JavaScript code could also be dynamically inserted into the DOM of a web site by using the `document.write` method. Several other possibilities to add JavaScript dynamically exist and catching all of them seems like a very difficult task.
2. Even if one is able to get the entire JavaScript code, there is still a problem when applying static analysis. This technique is not capable of distinguishing whether the JavaScript code will be used or not. The issue here is the fact that it's not sure that all parts of the code are actually executed. It is thinkable that only parts of them are used. Static analysis however will also extract features of unused parts of the code.
3. Building the AST and traversing it afterwards in search of relevant features can be pretty CPU-intensive, especially if there is a lot of JavaScript code embedded in the web site. This might lead to problems concerning the requirement to not delay the page load.

This Section showed that building ASTs from JavaScript code and using them to extract relevant, sough-after features, is a legitimate possibility. With Acorn and Esprima, two different libraries have been presented that make transforming JavaScript into ASTs incredibly easy. Moreover, evaluating the ASTs is a simple task, which can be achieved by using the estraverse library.
The next Section depicts another possibility to find out features: overwriting JavaScript objects.

Overwriting JavaScript

This Section shows another possibility to log accesses of FunctionCalls and MemberExpressions of a web site: overwriting JavaScript functions and members. At first, JavaScript's prototype nature will be explained, since this is fundamentally important for understanding this technique.

JavaScript's Prototype JavaScript is an object-oriented language. However, it implements inheritance not with classes, as most of the other object-oriented languages like Java or C++ do, but with prototypes. There exist several methods to create an object. The most simple one, an `object literal`, is shown in 5.4.

Listing 5.4: Person object in JavaScript.

```
1 var Person = {
2     name: 'Person'
3 }
```

Each time this object literal is evaluated, a new object is created and initialized. Another possibility to create objects is the keyword `new`. This keyword is always followed by a function invocation. Using a function like that is referred to as `constructor`, which also creates and initializes a new object. For native types, constructor functions are already defined, as depicted in Listing 5.5.

Listing 5.5: Creating native JavaScript objects with constructor functions.

```
1 var obj = new Object();
2 var arr = new Array();
```

Of course, it is also possible to define own constructor functions for initializing objects. This will be shown later in this thesis.

Now, the focus is on JavaScript's prototype nature. Each object in JavaScript is connected to another JavaScript object, its `prototype`. The created object inherits properties from its prototype. Objects, built with object literals as presented in Listing 5.4 do have the same prototype: `Object.prototype`. This means that they inherit all properties from this prototype, for example the methods `toString()` and `toSource()`. If constructor functions are used to create objects, the prototype is the prototype of the constructor function. For example, creating an object with `var arr = new Array();` results in the prototype of `arr` being `Array.prototype`, since `Array` is the constructor function.

There also exists a third possibility to create objects: the `Object.create()` method. This method's first argument is the prototype that should be used for the newly created object. Listing 5.6 shows, how this method can be utilized.

Listing 5.6: Creating objects with the Object.create() Method.

```
var proto = {
    forename: 'Thomas',
    surname: 'Mueller'
};
// Create an object with 'proto' as prototype
var obj1 = Object.create(prototype);

// obj inherited the 'forename' and 'surname' properties from the prototype
console.log(obj.forename);          //  yields 'Thomas'
console.log(obj.surname);           //  yields 'Mueller'
```

It's important to stress that `Object.prototype` is one of the few objects that does not have a prototype and therefore does not inherit any properties.
As already mentioned, it is also possible to write own constructor functions. This is shown in Listing 5.7. Here, a constructor function for `Person` is created. This is usually referred to as a class in JavaScript. Using this function with the `new` keyword creates new objects with their prototype being set to `Person.prototype`.

Listing 5.7: Person class in JavaScript.

```
function Person(name) {
    this.name = name;
}

var p = new Person('Tom');
```

Since all objects of this class inherit the properties from `Person.prototype`, it is possible to change this prototype. Listing 5.8 shows, how this can be achieved.

Listing 5.8: Changing the Person.prototype.

```
Person.prototype = {
    sayHello: function() {
        console.log("Hello from " + this.name);
    },

    sayGoodbye: function() {
        console.log("Goodbye from " + this.name);
    }
}

var p = new Person("Tom");

p.sayHello();                       // yields "Hello from Tom"
p.sayGoodbye();                     // yields "Goodbye from Tom"
```

The changes in the Person.prototype are reflected in every instance of this class.

Another important point when talking about prototypes is property access. As already stated, objects inherit all properties from their prototype. But what happens, when a property of an object is accessed that this object does not have. For example, let's assume to access the non-existent property `sayGoodMorning()` on a Person object. At first, the objects tries to query its own properties in search for sayGoodMorning(). If this property can not be found, its prototype will be queried. If the prototype does not have this property, again the prototype of the prototype will be queried. This procedure is going on until the property is found or an object with the property being `null` is queried. The linked list of prototypes of objects is called `prototype chain`.

Getting the prototype of an object is quite easy and can be achieved by using two different ways [60]:

1. Each object created with a constructor function has an property called `__proto__`. Although this is a nonstandard technique, it can be used to get the object's prototype object.

2. The `Object.getPrototypeOf()` method that has been introduced with ECMAScript 5, is another possibility to get the prototype. It yields the same result as the __proto__ object.

Last but not least, it's necessary to explain the difference between `prototype` and `__proto__`, since this is something that is often misunderstood. `prototype` is the prototype that is used when a new object is created. `__proto__` is an internal property of a object that points to its prototype. So consequently, the Listing 5.9 yields true:

Listing 5.9: Comparing __proto__ and prototype.

```
1 var p = new Person("Tom");
2
3 p.__proto__ === Person.prototype;
```

To come back to the original point, this knowledge can now be applied to find out, what features are used on a web site. Therefore, another fundamental technology of the JavaScript language is exploited. With the `Object.defineProperty(obj, prop, descriptor)` method, it is possible to overwrite existing properties of objects. The method expects the following arguments [61]:

- *obj*: Denotes the object on which to define the property.
- *prop*: The name of the property to be altered or defined.
- *descriptor*: The descriptor for the altered or defined property.

Descriptor is an object of itself created by the object literal notation. It has the following keys to be defined.

- *configurable*: true, if the behavior of this property may be changed, for example make it non-writable. Default: false
- *enumerable*: true, if the property should appear when enumerating the properties of this object. Default: false
- *value*: The value of the property. Default: undefined
- *writable*: true, if the value of this property can be changed by assigning a new value. Default: false
- *get*: Function conducing as the getter of this property. When there is no getter, it is `undefined`. The return value of this function will be used as the value of the property. Default: undefined
- *set*: Function conducing as the setter for this property, or `undefined`, if there is no setter. The only argument of this function is the new value of the property. Default: undefined

There exist two types of descriptors: *data descriptors* and *accessor descriptors*. The first ones are used to define a property having a value and being writable or not writable. The second one defines the property with a getter-setter pair of functions. A descriptor must either be a data descriptor or an accessor descriptor, not both.

Let's show this in a quick example. Listing 5.10 demonstrates, how the property `sayHello` is overwritten by using an accessor descriptor. Since this is done on the prototype object, every instance of the class Person now has this modified `sayHello` method. Since `configurable` is set to false, this means that this property will not be changeable in the future. Trying so will result in an exception.
The second part in this listing shows, how an existing instance is modified. This is realized by exploiting a data descriptor, which leads to its `name` property being overwritten to yield a new name.

Listing 5.10: Overwriting properties with Object.defineProperty.

```
1 Object.defineProperty(Person.prototype, "sayHello", {
2     configurable: false,
3     get: function() {
4         return function() {
5             return "Modified!";
6         }
7     }
8 });
9
```

```
10 p.sayHello();                        // yields "Modified"
11
12 // Create a new Person with name max.
13 var x = new Person("Max");
14
15 // Overwrite the 'name' property to yield a new name
16 Object.defineProperty(x, "name", {
17     value: "My name has been overwritten"
18 });
19
20 x.name;                              // yields "My name has been overwritten"
```

Looking back on the relevant features, it is striking that all of them are either simple accesses of properties (`plugins`) of certain objects (`navigator`) or functions, called on objects. With the functionality mentioned before, it is possible to overwrite those functions. However, their original functionality should be the same; it is enough to just log that this property has been accessed. Listing 5.11 demonstrates, how `navigator.plugins` could be overwritten, but also maintain its original functionality.

Listing 5.11: Overwriting properties with Object.defineProperty and maintaining its original functionality.

```
1  var property = "plugins";
2  var object = navigator;
3  var original = object[property];
4  Object.defineProperty(
5      object,
6      property,
7      {
8          enumerable: true,
9          configurable: false,
10         get: function() {
11
12             console.log("navigator.plugins has been called!");
13
14             return original;
15         }
16     }
17 );
```

Overwriting the `navigator` object like that leads to every access of the `plugins` property being logged to the console. Of course, this is very simple example how this technique can be used. Nevertheless, this technique can be extended to log all accesses to relevant features. In the future work, this technique will be called *poisoning*.

The great advantage of this approach is that it's applicable in browser extensions and that only features that are actually accessed, are logged. This is a big gain compared to using the AST. However, there are also downsides of this technique, as shown below.

1. First of all, in order to ensure wide distribution of the system, it is necessary to implemented browser addons for each existing browser - Mozilla Firefox, Google Chrome, Apple Safari and so on. The problem here is that the JavaScript code used to poison existing objects and their according properties is not portable [47], which means that the extension has be be developed from scratch for each browser.
2. Secondly, trackers are able to detect such poisoning, for example by calling the `Object.getOwnPropertyDescriptor(object, property)`[4] method that returns the descriptor of this property, which can be used to check for getters or setters of the property. In Listing 5.12, the differences between the return values of the `Object.getOwnPropertyDescriptor(object, property)` method of a poisoned and a non-poisoned property are demonstrated. It is clear that the property descriptor of a poisoned property can easily be identified.

Listing 5.12: Exemplary use of the Object.getOwnPropertyDescriptor() method.

```
1 // Get the property descriptor from a poisoned property.
```

[4] `https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Object/getOwnPropertyDescriptor`

```
>> Object.getOwnPropertyDescriptor(navigator, "plugins")
>> Object { configurable: false, enumerable: true, get: .get(), set: undefined }

// Get the property descriptor for a non-poisoned property.
>> Object.getOwnPropertyDescriptor(navigator, "plugins")
>> undefined
```

To sum it up, two different approaches to log relevant features were presented: using the AST of JavaScript code or overwriting JavaScript properties. Both of them are suited to solve the task. Working with ASTs is a quite promising approach, because the AST can not only be used for this purpose, but also for several other means like scanning for specific patterns in the source code or even alter it. However, the disadvantages somehow outweigh the advantages. Especially the fact that catching the entire JavaScript code of web sites is extremely difficult to realize, is a huge downside.
Overwriting or "poison" JavaScript properties is also a good candidate to solve the objective. In particular, its simple implementation and it only logging features that really are accessed, have to be stressed out.

All in all, both variants were implemented in form of browser extensions. While the AST-based approach seemed quited promising at the beginning, it became clear that it is incredibly hard to get all JavaScript code of a web site using a browser extension. Moreover, not only features that are actually used by the web site have been logged, but anyone present in the JavaScript code. This is also a big problem, because it might lead to wrong classification. On the other hand, the approach to poison the relevant JavaScript objects worked out very well. The developed browser extension poisons the relevant JavaScript objects like navigator, screen and Date and logs calls to appreciable MemberExpressions and FunctionCalls. However, one challenge here was to find out, when the collected features should be sent to the server. The naive approach is to transmit them, when the `window.onload` event fires, because this is the time, when the entire page, including its content (images, css, scripts) has been finished loading [62]. However, as it turned out, this point in time is actually too early to use as the event that triggers the extension to send the collected features to the server. In fact, there are still JavaScript files that are evaluated after this event fires and features accessed by these scripts would than not be part of the transmitted data. Therefore, it became apparent that it is necessary to wait a given time (30 seconds for example) before the features are sent to the server.

5.1.2. Data Collection

In the distributed approach used in this implementation, the sensors represent the clients. What's missing right now, is the server, which involves data collection and evaluation. These parts will be discussed in the following Sections.

From a technical point of view, the server is based on a web server using *flask*[5]. Flask is a BSD licensed microframework for Python. The web server exposes three APIs:

- `/collect`: Expects a POST request and stores the input in the database in order to use it as training data.
- `/classify`: Also awaits a POST request with the data having the same format. But the input is classified using the classifier of the logic/evaluation layer. The result (True/False) is sent back to the client.
- `/overview`: Called via GET request. Returns an overview of the current training data available.

The web server utilizes a *MySQL*[6] database. The scheme of this database is shown in Figure 5.2. It is slightly different to the one presented in Figure 4.3.

The reason for this is that only JavaScript-based features were collected and therefore only two different types of features are taken into account: MemberExpressions and FunctionCalls. Moreover, for each of those types, one table has been created in order to be able to use the ORM library *peewee*[7]. The *feature_type* table is consequently dropped. Furthermore, two tables are necessary to realize the many-to-many relationship

[5] `http://flask.pocoo.org/`
[6] `https://www.mysql.de/`
[7] `https://peewee.readthedocs.org`

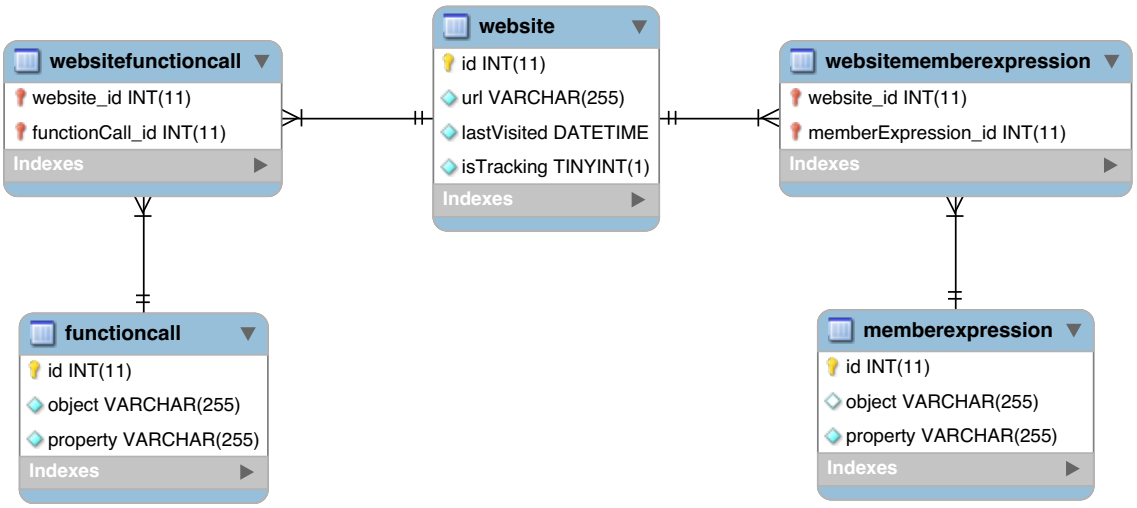

Figure 5.2.: Database scheme used in the prototype implementation.

between web sites and FunctionCalls as well as web sites and MemberExpressions. In Figure 5.2, the resulting database scheme used in the prototype implementation is depicted.

5.1.3. Logic/Evaluation

This Section focuses on the logic/evaluation layer. Section 5.1.3 will show how a suitable training data set has been created. After that, a popular library involving several detection mechanisms is presented in Section 5.1.3. Lastly, Section 5.1.3 will concentrate on how the collected data has to be preprocessed in order to be able to use it for training or to classify it.

Building Training Data

The detection mechanisms presented in 2.4 have in common that they need to be trained to be capable of classifying new input. Therefore that data collection layer has to provide training data. As already discussed, there doesn't exist a list of web sites that are known to track their users. Fortunately, Gunes Acar, one of the author of [40], had the same problem and was able to create a list with web sites that are guaranteed to do Canvas-Fingerprinting. At first, its correctness has been verified using the Chameleon (see Section 3.2.1) browser extension. After that, each web site of this list has been visited with a browser that had the sensor developed in this thesis (which is a browser extension), installed. Therefore, when visiting a web site with this browser, the sensor sends the relevant features to a specific interface provided by the server. The features for each web site are then stored in the database. The list also contains web sites that are considered to not track their users. Their features have also been extracted and saved in the data collection layer. The result is a set of training data, consisting of tracking web sites and not tracking web sites, along with their features, which is a perfect fit for training classifiers.

In order to automate this process of getting training data, Selenium has been used. With this, instrumenting the browser to visit one web site after another can be achieved. Listing 5.13 shows, how Selenium can be used to visit one web site after another with a Firefox browser that has installed a specific extension.

Listing 5.13: Visiting a list of web sites with a browser containing a JavaScript sensor.

```
1 from selenium import webdriver
2 import time
3
4 TIMEOUT = 15
5
```

```
6  fp = webdriver.FirefoxProfile()
7  fp.add_extension('/path/to/browser_extension/firevest/@firevest-0.0.1.xpi')
8
9  browser = webdriver.Firefox(firefox_profile=fp)
10
11 with open('web sites.txt', 'r') as file:
12     sites = file.readlines()
13
14 number_of_sites = len(sites)
15 count = 1
16
17 for site in sites:
18     print("Visiting {}".format(site.rstrip()))
19     print("This is {} of {}".format(count, number_of_sites))
20     browser.set_page_load_timeout(30)
21     try:
22         browser.get(site)
23     except:
24         print("Timeout when visiting: " + site)
25
26     # Wait 60 seconds before visiting the next web site
27     time.sleep(60)
28
29     count += 1
30
31 browser.close()
```

Scikit-Learn

The detection mechanisms presented in Section 2.4 are among the most popular ones, because they are rather easy to implement, but still yield very good results and are great in terms of performance. Therefore, it is not surprising that they are implemented in several libraries for numerous programming languages. For example, just for Bayesian classifiers exist a wide range of implementations like Scala[8], JavaScript[9] or the more traditional Java[10]. Another common possibility is to focus on the R programming language (`http://www.r-project.org/`) that was explicitly designed for statical computing and consequently implements a wide range of different detection mechanisms. However, R has been developed with a focus on statical computing and graphics [63]. Therefore, its other areas of application are somewhat limited.
Another very interesting project in the field of detection mechanisms is the Python library *scikit-learn* [64]. It is built upon the well-known libraries *NumPy*, *SciPy* and *matplotlib* and is completely open source. It provides a number of diverse tools from the fields of data mining and data analysis, including [64]:

- Classification (nearest neighbors, support vector machines, random forest, extra trees, ...)
- Regression (ridge regression, Lasso, ...)
- Clustering (k-Means, spectral clustering, mean-shift, ...)
- Dimensionality reduction (feature selection, ...)
- Model selection (grid search, cross validation, ...)
- Preprocessing (feature extraction, preprocessing, ...)

It becomes clear that scikit-learn offers everything the logic/evaluation layer requests. Moreover, Python is a general-purpose programming language and using it assures that every part of the evaluation layer can be implemented without boundaries imposed by the programming language.

The prototype developed in this thesis uses the implementation of the DecisionTree, Naïve Bayes and Extra-Trees of scikit-learn.

[8] `https://github.com/arnaudleg/naive-bayes-classifier-scala`
[9] `https://github.com/ttezel/bayes`
[10] `https://github.com/ptnplanet/Java-Naive-Bayes-Classifier`

Data Preparation

Scikit-learn requires the data to have a specific format to be of use. Each web site has to be transformed into this format, so that scikit-learn is capable of understanding the data. In the following, it will be explained how the data stored in the data collection layer can be transformed into the required format.

The relevant features defined in Section 4.3.1 can be arranged in a fixed order. Let's assume the fixed order of the feature is the same they appear in Table 4.1. This ordering is shown in Listing 5.14.

Listing 5.14: Fixed ordering of relevant features.

```
ordered_features = [
    FunctionCall(object='HTMLCanvasElement', property='toDataURL'),
    FunctionCall(object='HTMLCanvasElement', property='getContext'),
    FunctionCall(object='CanvasRenderingContext2D', property='fillText'),
    FunctionCall(object='CanvasRenderingContext2D', property='strokeText'),
    FunctionCall(object='CanvasRenderingContext2D', property='getImageData'),
    FunctionCall(object='WebGLRenderingContext', property='getParameter'),
    FunctionCall(object='WebGLRenderingContext', property='getSupportedExtensions'),
    FunctionCall(object='Date', property='getTimezoneOffset')
    MemberExpression(object='navigator', property='plugins'),
    MemberExpression(object='navigator', property='userAgent'),
    MemberExpression(object='navigator', property='language'),
    MemberExpression(object='navigator', property='cpuClass'),
    MemberExpression(object='navigator', property='platform'),
    MemberExpression(object='navigator', property='doNotTrack'),
    MemberExpression(object='screen', property='height'),
    MemberExpression(object='screen', property='width'),
    MemberExpression(object='screen', property='colorDepth'),
    MemberExpression(object='window', property='sessionStorage'),
    MemberExpression(object='window', property='openDatabase'),
    MemberExpression(object='window', property='localStorage'),
    MemberExpression(object='window', property='ActiveXObject'),
    MemberExpression(object='window', property='indexedDB'),
    MemberExpression(object='window', property='devicePixelRatio'),
    MemberExpression(object='window', property='innerWidth'),
    MemberExpression(object='window', property='innerHeight'),
    MemberExpression(object='HTMLElement', property='offsetHeight'),
    MemberExpression(object='HTMLElement', property='offsetWidth'),
]
```

Consequently, each web site can be transformed into a list of boolean values, where each value states, if the feature at this index is present on the web site or not. This representation of a web site will be called *binary vector*. So if for example the value at index 0 in the binary vector of a web site is 1, the feature at index 0 in the ordered feature list (`FunctionCall(object='HTMLCanvasElement', property='toDataURL')`) is present on the web site.

Let w be a web site with a list of features denoted as $features$, which is a property of w. The binary vector for w is then built[11] as shown in Listing 5.15.

Listing 5.15: Pseudo code for building the binary vector of a web site.

```
def create_binary_vector(w):
    relevant_features = get_ordered_features()
    vector = list()
    for index, feature in relevant_features:
        if feature in w.features:
            vector[index] = 1
        else:
            vector[index] = 0
    return vector
```

Let's assume, the features of a web site are the ones shown in Listing 5.16.

[11] This code listing is written in pseudo code being loosely based on Python syntax.

Listing 5.16: Exemplary feature set of a fictional web site.

```
features = [
    FunctionCall(object='HTMLCanvasElement', property='toDataURL'),
    FunctionCall(object='HTMLCanvasElement', property='getContext'),
    FunctionCall(object='CanvasRenderingContext2D', property='fillText'),
    FunctionCall(object='Date', property='getTimezoneOffset')
    MemberExpression(object='navigator', property='plugins'),
    MemberExpression(object='navigator', property='userAgent'),
    MemberExpression(object='navigator', property='language'),
    MemberExpression(object='HTMLElement', property='offsetHeight'),
    MemberExpression(object='HTMLElement', property='offsetWidth'),
]
```

Listing 5.15 shows, how the binary vector for this fictional web site can be calculated. The result is presented in Listing 5.17.

Listing 5.17: Binary vector of the fictional web site.

```
binary_vector = [1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    1, 1]
```

Scikit-learn heavily relies on this representation of data. In fact, the training data also has to be structured in that way. Training the three detection mechanisms presented in Section 2.4 is done by calling the `fit(X, y)` method [65, 66, 32], which expects two values:

- `X`: A list of lists, where each inner list is a binary vector of a web site.
- `y`: A list containing classes. In this case, only boolean values are considered, since the only classes are tracking and not tracking. Therefore, this list contains boolean values. The index of the boolean value denotes the class of the binary vector in `X` at the same index. For example, if `y[0] == 1`, the web site with the binary vector at `X[0]` is marked as tracking the users.

Binary vectors play an essential role in scikit-learn, because they are not only used for training but also for classifying new input. Therefore, if a new web site has to be classified using one of the trained classifiers, this web site's binary vector is generated and passed in the classifier's `predict(X)` method as input. The return value of this method call is a class; tracking or not-tracking.

5.2. Evaluation and Methodology

The results that were collected with the prototype system will be evaluated in this Section. Therefore, all tested classifiers will be presented. Moreover, four different test cases will be demonstrated and it will be explained, how they are designed. After that, the performance of every classifier for every test will be shown and analyzed.

For the purpose of evaluating the classifiers, it is inevitable to have a set of training data. Section 5.1.3 discussed in detail how the training data has been collected. This set constitutes the basis of the whole assessment process. The entire training data consists of two distinct sets: the one that involves all web site that are tracking their users (*tracking set*) and the one with web sites that are considered to not track their users (*not-tracking set*).

Overall, the training data consist of 82 different web sites, with 41 being marked as tracking and 41 as not-tracking. The tracking set has been collected with the help of Gunes Acar, who offered a list of web sites that are using Canvas-Fingerprinting. The not-tracking set is built of randomly selected sites among the Alexa Top 100. The whole set of training data is illustrated in Section A.3.

At the moment, there are four different classifiers implemented [65, 66, 32].

- Naïve Bayesian Classifier[12]

[12]The concrete implementation uses a Gaussian Naïve Bayes.

- DecisionTree
- ExtraTrees
- Aggregator

To measure the performance of a classifier, four different test have been designed. They differ in the web sites used for training and classifying and in their respective features. These are relevant for both training and classifying and a different set of features will result in a different outcome. At the beginning of each test, the set of training data is split into two separate sets: *test set* and *training set*. The employed strategy is called *EqualSplit*. Here, the tracking set and not-tracking set are each split into two equally sized parts, where the first part of each resulting set belongs to the training set and the second part to the test set. This splitting strategy makes sure that always the same web sites are used for training and for testing. This is necessary for comparing the results of the various classifiers. The resulting sets are depicted in Sections A.2 and A.1. As it becomes clear, there are 40 web sites used for training and 42 web sites are tested with the trained classifier.

Moreover, two different types of feature sets have been identified.

- With Canvas-Fingerprinting: This feature set contains all features presented in Table 4.1. Every one of them is used for training and classifying.
- Without Canvas-Fingerprinting: In this feature set, features that are considered to be highly relevant only for Canvas-Fingerprinting, are left out. This set is used to determine the success rate of the classifiers for Browser-Fingerprinting only.

Table 5.1 shows the features that are not considered in this feature set. The ignored ones are features that are specific to Canvas-Fingerprinting and not to Browser-Fingerprinting.

Table 5.1.: Features that are left out in the set without Canvas-Fingerprinting-related features.

Type	Object	Property
FunctionCall	HTMLCanvasElement	toDataURL
FunctionCall	HTMLCanvasElement	getContext
FunctionCall	CanvasRenderingContext2D	fillText
FunctionCall	CanvasRenderingContext2D	strokeText
FunctionCall	CanvasRenderingContext2D	getImageData
FunctionCall	WebGLRenderingContext	getParameter
FunctionCall	WebGLRenderingContext	getSupportedExtensions

After the training and test sets are created, the first one is used to train the classifier and the second one is used to evaluate its performance. Therefore, each web site from the test set is classified. After that, the result of the classification is compared with the predefined class. There are three different outcomes of this comparison.

- *Correct*: The result predicted by the classifier is the same as the predefined one.
- *False-Positive*: The classifier predicted the web site to track users, but in fact it is not.
- *False-Negative*: The classifiers predicted the web site to belong to the not-tracking set, but it is part of the tracking set.

To sum it up, one test is performed in the following order.

1. Split the training data into training set and test set using the splitting strategy.
2. Train the appropriate classifier using the training set.
3. Classify each web site of the test set with the trained classifier.
4. Compare the predicted class with the actual class of the web site.

Table 5.2 demonstrates all tests that are performed in this Section. As one can see, this results in a total of eight tests, two tests for each classifier. They differ in the feature sets, used for training and classifying.

Table 5.2.: Overview of all performed tests.

Classifier	Splitting Strategy	Canvas-Fingerprinting
Naïve Bayes	EqualSplit	✓
Naïve Bayes	EqualSplit	✗
DecisionTree	EqualSplit	✓
DecisionTree	EqualSplit	✗
ExtraTrees	EqualSplit	✓
ExtraTrees	EqualSplit	✗
Aggregator	EqualSplit	✓
Aggregator	EqualSplit	✗

The following Sections will present the outcome of the tests for each classifier. As already stated, two tests will be executed for each classifier, one with all features to be considered, and one with the Canvas-Fingerprinting-related ones ignored.

5.3. Measurements

5.3.1. Naïve Bayesian Classifier

All Features

This test considers all features. The results are shown in Table 5.3. As one can see, only two web sites from the test set are classified incorrectly. The web sites `http://www.imgur.com/` and `http://www.aliexpress.com` are classified as tracking, but, in fact, they are not tracking their users. Nevertheless, only two out of 42 are misclassified, which yields a success rate of 95%.

Table 5.3.: Bayes EqualSplit with all features.

URL	False-Positives	False-Negatives
`http://imgur.com/`	✓	
`http://www.aliexpress.com/`	✓	
Summary	2	0

Without Canvas-Fingerprinting-related Features

In this test, Canvas-Fingeprinting-related features are not taken into account. The outcome differs critically to the test shown in Section 5.3.1. This time, there are seven False-Positives and three False-Negatives, which equals a total of ten web sites not being misclassified. Table 5.4 illustrates the performance. Having misclassified ten out of 42 web sites results in a success rate of only 76%.

The increasing number in misclassified web sites is not surprising considering the features that are incredibly strong indicators for tracking are not considered here. Moreover, it strikes that the False-Positives of the last test presented in Section 5.3.1 are also misclassified here. This means that those two web sites can not be classified correctly although the features have changed. This leads to the assumption that misclassification is not related to the features that have been left out.

Table 5.4.: Bayes EqualSplit without Canvas-Fingerprinting-related features.

URL	False-Positives	False-Negatives
http://www.sohu.com/	✓	
http://imgur.com/	✓	
https://www.pinterest.com/	✓	
http://instagram.com/	✓	
https://www.paypal.com/de/webapps/mpp/home	✓	
http://www.aliexpress.com/	✓	
http://www.alibaba.com/	✓	
http://us.webnode.com/		✓
http://www.webnode.cz/		✓
http://www.pof.de/		✓
Summary	7	3

5.3.2. Decision Tree

The Decision Tree has been built with the following parameters [32]:

- criterion: *gini*
  This is the function that is used to measure the quality of a split (the splitting strategy).
  Options are *gini* (Gini Index) and *entropy* for (Information Gain).
- splitter: *best*
  Strategy utilized to choose the split at each node.
  Options are *best* (select the best split) and *random* (select the best random split).

The Decision Tree implementation of scikit-learn offers more parameters [32] than the ones listed above, but those were the only ones being considered. For the other parameters, the default values have been taken. Of course, optimizing the parameters might lead to an improved success rate of the classifier. However, due to the sheer amount of possibilities, this has not been done here. This is part of possible improvements and will be explained in Section 6.2.

All Features

This test considers all features to be relevant. Its outcome is extremely accurate, resulting in zero misclassified web sites and consequently yielding a success rate of 100%.

One great thing about Decision Trees is the fact that they can be visualized in order to ease interpretation. In Figure 5.3, one can see the visualized Decision Tree after it has been trained. This representation is great for understanding, how Decision Trees work.

Table 5.5.: Feature importances for Decision Tree with all features.

Feature	Importance
openDatabase	0.818181818182
fillText	0.181818181818

Interpreting this visualization is very straight forward. Each node consists of three parts: The first part is the criteria that is used to determine the next child node. Since only binary values are considered here, this means that if the feature mentioned is not present, its value is zero. If the condition yields true, the left child node is taken, else the right child node. The second part of the node shows the value calculated by the splitting

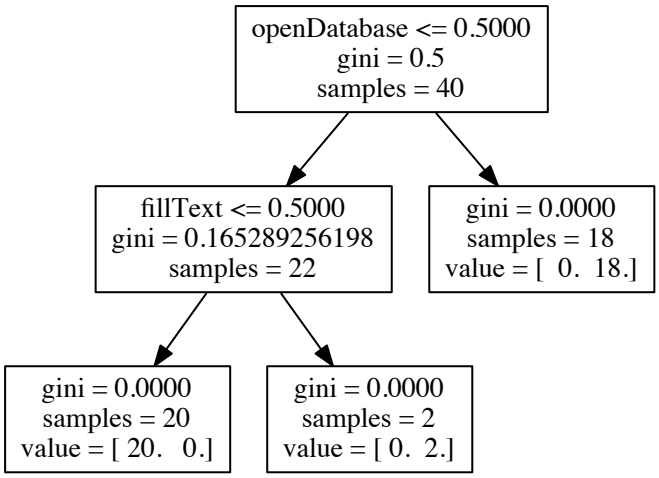

Figure 5.3.: Decision Tree visualization for Equal-Split with all features.

strategy used. The part labeled with "samples" shows the total number of web sites of this node. Leaf nodes however do not contain the condition but instead, they denote the classes. This is done in the part labeled with "value". This part shows, how many web sites belong to what classes. Again, only boolean data is considered. Therefore, the value is an array of two, with the number at the index 0 representing the total count of web sites classified as not-tracking. The number at index 1 denotes the total count of web sites classified as tracking. For example, *value = [20. 0.]* means that 20 web sites are not tracking and zero are classified as tracking.
Having this knowledge, the image illustrated in Figure 5.3 is simple to interpret. The first split is done using the feature `openDatabase`. The training set is consequently divided into two parts: one in which all web sites use this feature and one with all web sites not using this feature. The choice of `openDatabase` is not surprising, because this is one of the features, almost every tracking web site uses. Therefore, it can divide the training set in two almost equally sized subsets. The second split is done in the set containing `openDatabase` and using the feature `fillText`. Again, this results in two subsets with web sites using this feature and web sites not using this feature. To sum it up, web sites accessing openDatabase are immediately classified as tracking, else the next feature of importance is fillText. If this feature is present on the web site, it is considered to track, otherwise, its class is not-tracking.

Scikit-learn provides a functionality to print out the feature importances. To preserve the overall view, only features with an importance unequal to zero are listed. Table 5.5 shows the feature importances for this test case. The features depicted here are also the ones used as splitting features.

Without Canvas-Fingerprinting-related Features

Here, the relevant features do not contain the Canvas-Fingerprinting-related ones. The result reminds one of the Naïve Bayesian classifier's outcomes. The total number of misclassified web sites is increasing if not all features are considered. The result of this test is shown in Table 5.6. As one can see, there is a total of two False-Positives and four False-Negatives, which sums up to six misclassified web sites. This still means a success rate of 86%. Interestingly, the Decision Tree has a higher number of False-Negatives - it tends to predict web sites to not track, although they are tracking. This differs from the Naïve Bayesian classifier, who yields a higher number of False-Positives.

The visualized Decision Tree is illustrated in Figure 5.4; its feature importances as shown in Table 5.7. Compared to the one considering all features, this tree became more complex and more splits were necessary.

Table 5.7 shows the feature importances of this tree. The best feature is still `openDatabase` with an importance of ≈ 0.82. This is a very high value, especially when compared with the importances of the next features `devicePixelRatio` (≈ 0.08), `localStorage` (≈ 0.07) and `language` (≈ 0.04).
Since there are now four splits necessary, the tree becomes bigger and more complex. Figure 5.4 shows exactly,

Table 5.6.: Results for the Decision Tree without Canvas-Fingerprinting-related features.

URL	False-Positives	False-Negatives
`https://www.pinterest.com/`	✓	
`http://stackoverflow.com/`	✓	
`https://robertsspaceindustries.com/`		✓
`http://us.webnode.com/`		✓
`http://www.webnode.cz/`		✓
`http://www.pof.de/`		✓
Summary	2	4

Table 5.7.: Feature importances for DecisionTree without Canvas-Fingerprinting-related features.

Feature	Importance
openDatabase	0.818181818182
devicePixelRatio	0.0761904761905
localStorage	0.0666666666667
language	0.038961038961

how the features are taken as splitting features one after another, creating the final Decision Tree.

5.3.3. ExtraTrees

The following parameters have been used in the tests [66]:

- n_estimators: *20*
  This parameter determines the number of trees.
  Default value: 10
- criterion: *gini*
  Again, this specifies the splitting strategy.
  Options are *gini* (Gini Index) and *entropy* for (Information Gain).
- max_features: *auto*
  Determine the number of features that are considered when the best split is searched for.
  Options are:
  - integer value: max_features are considered at each split.
  - float value: max_features is a percentage; at each split, $int(max\_features \cdot n\_features)$[13] are considered.
  - "auto": $max\_features = \sqrt{n\_features}$
  - "sqrt": $max\_features = \sqrt{n\_features}$
  - "log2": $max\_features = \log_2 n\_features$
  - None: $max\_features = n\_features$

The scikit-learn implementation of the ExtraTrees classifier provides a lot more parameters [66], but due to the incredible high amount of possibilities, those are set to their default values. Tweaking them can however improve the detection rate of the classifier. This will also be mentioned in Section 6.2.

[13] `n_features` denotes the total number of features used for building the classifier.

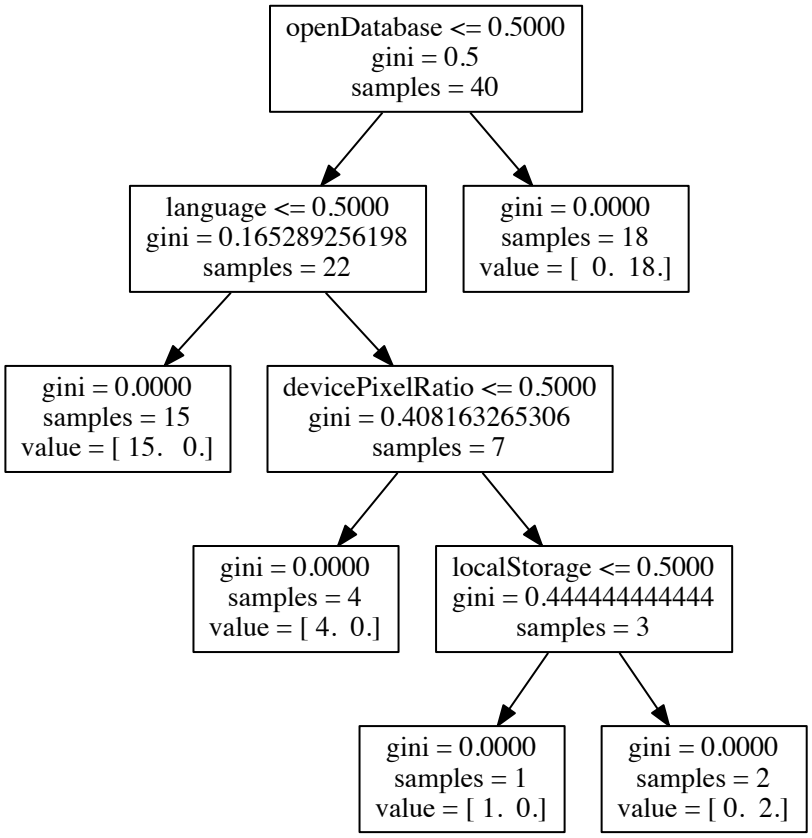

Figure 5.4.: Decision Tree visualization for Equal-Split without Canvas-Fingerprinting-related features.

It is important to mention that due to the randomized creation of ExtraTrees, is is not guaranteed to get the same results for a test, even if the same web sites are used for training. The reason for this lies in the way ExtraTrees are built up. They are effectively not just one tree but a complete forest, with each tree being built with randomly selected splitting features. Because of this randomization, the resulting forest of trees is almost always different, which might lead to varying outcomes.

All Features

This test considers all features and the training data are split using the Equal-Split strategy. The results are really good with only one web site being misclassified. The outcome is illustrated in Table 5.8. With only one False-Positive, the success rate is 98%. This means that the ExtraTrees classifier is slightly behind the DecisionTree in this test.

Table 5.8.: Results for ExtraTrees with all features.

URL	False-Positives	False-Negatives
`http://www.microsoft.com/de-de/default.aspx`	✓	
Summary	1	0

Without Canvas-Fingerprinting-related Features

This is where things get usually more interesting. As it has been the case in all of these tests before, the number of misclassified web sites is higher. This is also true for the ExtraTrees classifier. However, this test produces five misclassified web sites and all of them are False-Negatives. Nevertheless, a success rate of 88% is still a very strong result.

Table 5.9.: Results for ExtraTrees without Canvas-Fingerprinting-related features.

URL	False-Positives	False-Negatives
`https://robertsspaceindustries.com/`		✓
`http://us.webnode.com/`		✓
`http://www.webnode.cz/`		✓
`http://www.cheatsheet.com/`		✓
`http://www.pof.de/`		✓
Summary	0	5

An interesting aspect is that, just like it has been the case with DecisionTrees, ExtraTrees also seem to have problems to detect tracking web sites when Canvas-Fingerprinting-related features are left out. In fact, all False-Negatives of DecisionTrees appear in the False-Negatives of ExtraTrees. Therefore, the ExtraTrees does not yield any False-Positives like the DecisionTree which makes him overall superior to the DecisionTree, at least in this test.

5.3.4. Aggregator

This classifier is not based on a new technique in the field of Data Mining or similar topics, but is rather a combination of the three classifiers already presented. The idea is to combine the results of each of those classifiers and mix them together. The final result is the one that has the absolute majority. This means that for example, if the DecisionTree and ExtraTrees predict a web site to track, but the Naïve Bayesian classifier states that it is not tracking, the Aggregator will predict the web site to track the users because this result is in the majority. Right now, every classifier is considered to be equally important. Another possibility is to weigh classifiers, such that one classifier is more important than the others. This approach seems reasonable, if it turns out that one classifier is far superior to the others.

All Features

This test utilizes all features. There are no misclassified web sites in this test. The reason for this is that all three core classifiers performed really well and the Aggregator is able to adapt to this great performance. Although the ExtraTrees yielded one False-Positive and the Naïve Bayes produced two False-Positives, the Aggregator was still able to correctly classify all webs sites, because of it taking the majority result of the core classifiers. This means a success rate of 100%.

Without Canvas-Fingerprinting-related Features

Again, this is the more interesting test. The Canvas-Fingerprinting-related features are not taken into account. The outcome of the Aggregator is shown in 5.10.

Due to the nature of this classifier, the outcome has actually been clear even before the test has been executed. When a web site is misclassified by at least two of the other classifiers, the Aggregator will also fail to classify it correctly. This is exactly what happens for example to `https://robertspaceindustries.com/`.

Table 5.10.: Aggregator EqualSplit without Canvas-Fingerprinting-related features.

URL	False-Positives	False-Negatives
https://www.pinterest.com/	✓	
https://robertsspaceindustries.com/		✓
http://us.webnode.com/		✓
http://www.webnode.cz/		✓
http://www.pof.de/		✓
Summary	1	4

Although the Naïve Bayes is able to predict the correct class, the other two classifiers fail in doing so. Consequently, the Aggregator also fails. To sum it up, the Aggregator fails in five cases, where one web site is a False-Positive and four web sites are False-Negatives. Five misclassified web sites out of a total of 42 web sites leads to a success rate of 88%.

6. Discussion

The last Chapter presented the measurements of the classifiers. Section 6.1 will discuss those measurements and evaluate them in detail, beginning with differences in the training data. Moreover, the outcomes of all classifiers using the EqualSplit strategy will be compared with each other in Section 6.1.2. Furthermore, Section 6.1.3 will illustrate a second splitting strategy, which is used to measure the quality of the results of the EqualSplit. In Section 6.2, possible improvements to enhance the detection rates of the classifiers will be elucidated.

6.1. Discussion and Interpretation

6.1.1. Differences in the Training Data

As already mentioned, the system relies on the assumption that web sites from the tracking set and from the not-tracking set use different features. This proposition will be tested by examining what features are used in the tracking and in the not-tracking sets. Table 6.1 presents all features[1] along with the number of occurrences in the respective set.

This overview is really interesting and allows worthwhile conclusions at a first glance. Firstly, the features `toDataURL`, `getContext`, `fillText`, `getImageData`, `getParameter` and `getSupportedExtensions`, which all happen to be FunctionCalls, are almost exclusively found on web sites from the tracking set. Moreover, these features are considered to be only relevant in Canvas-Fingerprinting. Interestingly, the feature `strokeText` is not used on any web site at all. Since this seemed to be suspicious, it has been verified that the sensor correctly detects the usage of this feature, in order to rule out the possibility that this is the sensor's fault. There are other interesting features almost exclusively used in the tracking set, for example `openDatabase`, `indexedDB` or `doNotTrack`. The presence of those features seems to be a strong indicator that the web site is tracking their users. Since a lot of the strong features are almost exclusively related to Canvas-Fingerprinting, a second test with these features being ignored is executed. This should shed light on the classifier's capability to detect Browser-Fingerprinting.

However, some features seem to be no hint for the classification of the web site. For instance, the features `userAgent`, `height`, `width` or `ActiveXObject` are used in both sets frequently. This fact leads to the assumption that the presence of them is not a decisive factor in the classification of the web site.

6.1.2. EqualSplit Results

Section 5.3 demonstrated the outcomes of the tests using EqualSplit. To summarize, the overall performance of the different classifiers for all tests performed are presented in Table 6.2.

As this Table shows, the classifiers really excel when all features were taken into account. Especially the ExtraTrees and Decision Tree performed really well and were able to predict the right class for all 42 tested web sites or yielded only one misclassified web site. But also the Naïve Bayesian classifier achieved results close to the two tree-based classifiers with only two False-Positives.

If Canvas-Fingerprinting-related features were removed, the number of misclassified web sites increased for all classifiers. Nevertheless, this behavior is absolutely understandable because those features are the ones

[1] For simplicity reasons, the `object` of features are left out. The interested reader is advised to have a look at Table 4.1 which shows the features with both, object and property.

Table 6.1.: Occurrences of features in the tracking and not-tracking sets.

Feature (property only)	Tracking occurrences (41 in total)	Not Tracking occurrences (41 in total)
toDataURL	39	0
getContext	40	7
fillText	39	0
strokeText	0	0
getImageData	2	0
getParameter	2	0
getSupportedExtensions	2	0
getTimezoneOffset	40	16
plugins	41	26
userAgent	41	37
language	41	19
cpuClass	34	3
platform	41	25
doNotTrack	35	3
height	41	30
width	41	30
colorDepth	41	26
sessionStorage	34	13
openDatabase	29	0
localStorage	38	26
ActiveXObject	31	31
indexedDB	29	0
devicePixelRatio	23	15
innerWidth	28	18
innerHeight	28	21
offsetHeight	40	30
offsetWidth	39	30

which are almost guaranteed to be only used by web sites that are tracking their users. The reason for this lies in the way Canvas-Fingerprinting functions. If someone wants to apply Canvas-Fingerprinting, he has only very few options to do so. It is necessary to use the feature `toDataURL`, which enables the extraction of the data from the canvas. Also, the `getContext` method needs to be utilized, because it provides a context element which can be used for drawing. These very strong dependencies on certain features lead to them being an incredibly strong indicator for Canvas-Fingerprinting. The Decision Tree shown in Figure 5.3 elucidates this. The second splitting feature right after `openDatabase` was `fillText`. Web sites which are using both of them are automatically considered to track their users.

Browser-Fingeprinting is a more general technique. The difference to Canvas-Fingerprinting lies in the features. The reason for it being not as accurately classified as Canvas-Fingerprinting is that there are no completely typical features for it. However, there exist several features, which turned out to be almost exclusively used on tracking web sites but are not considered to be related to Canvas-Fingerprinting, for example `openDatabase` and `indexedDB`. The presence of them is a good indication, but it alone is not sufficient to classify a web site as tracking. The problem is that a lot of features that can be used in Browser-Fingerprinting are totally legit and are in fact used by a lot of web sites for benign reasons. For example, calls to the `screen.height` and `screen.width` MemberExpressions are no hint for tracking or not-tracking. This one is frequently used for determining the screen resolution, which is necessary to display the web site correctly. This dual use makes identifying the presence of Browser-Fingerprinting rather complex. Still, the classifiers performed really well, even when strong Canvas-Fingerprinting-related features were left out. Especially the ExtraTrees and Aggregator produced success rates of 88%, even in the more difficult test with not

Table 6.2.: Summary of the performed tests using EqualSplit.

Classifier	Test	False-Positives	False-Negatives	Correct	Correct (in %)
Aggregator	With Canvas	0	0	42	100%
	Without Canvas	1	4	37	88%
Bayes	With Canvas	2	0	40	95%
	Without Canvas	7	3	32	76%
Decision Tree	With Canvas	0	0	42	100%
	Without Canvas	2	4	36	86%
ExtraTrees	With Canvas	1	0	41	98%
	Without Canvas	0	5	37	88%

all features being considered. This is a clear sign that detecting Browser-Fingerprinting is possible.

The web pages `http://www.pof.de`, `http://us.webnode.com/` and `http://www.webnode.cz/` are really interesting ones, because those were misclassified by all classifiers when Canvas-Fingerprinting-related features were not taken into account. This is worth examining in detail. Obviously, the last two pages seem to be extremely similar, judging from the URL and as Table 6.3 demonstrates, they do in fact use the same features.

Table 6.3 shows all features of the three web sites of interest. In the following, it will be analyzed, why those web sites are misclassified. The easiest way to do this is by using the Decision Tree classifier, because of the possibility to visualize it. The visualization helps in comprehending the classification process. This is a great advantage to the other classifiers. Using the Decision Tree classifier demonstrated in Figure 5.4, the classification of the web page `http://www.pof.de` is performed.
The first feature of interest is `openDatabase`. The examined web site does not utilize this feature, so consequently the left child node is taken. Here, the relevant feature is `language`. This feature is in fact used, which leads to the right child node being visited next. In this node, the `devicePixelRatio` is the feature of interest. Since it is not present on the web site, the left child node is picked next. This is actually a leaf node, which results in the class of the page being *not-tracking*.
As Table 6.3 shows, the other two web site are similar concerning the above mentioned features. This means that all those web sites are misclassified in the Decision Tree, because of the combination of the features `openDatabase`, `language` and `devicePixelRatio`.

This example shows the difficulties when classifying Browser-Fingerprinting for at least three misclassified pages. Unfortunately, the other classifiers - Naïve Bayes and ExtraTrees - are more complex to analyze, because they don't provide such a helpful visualization. The ExtraTrees classifier for example consists of a whole set of such Decision Trees.

In summary, the Aggregator can be considered the best classifiers among all tested ones with a success rate of 100% for Canvas- and 88% for Browser-Fingerprinting. The second best ones were ExtraTrees and Decision Trees, whose results are almost as good as the ones from the Aggregator. Especially the Decision Tree's performance is somewhat surprising, considering the fact that it is an incredibly simple classifier. Naïve Bayes was also very good when all features were taken into account, but showed difficulties with detecting Browser-Fingerprinting only.

Table 6.3.: Feature of the examined web sites.

Feature	http://www.pof.de	http://us.webnode.com/	http://www.webnode.cz/
getTimezoneOffset	✓	✓	✓
plugins	✓	✓	✓
userAgent	✓	✓	✓
language	✓	✓	✓
cpuClass	✗	✗	✗
platform	✓	✓	✓
doNotTrack	✗	✗	✗
height	✓	✓	✓
width	✓	✓	✓
colorDepth	✓	✓	✓
sessionStorage	✗	✗	✗
openDatabase	✗	✗	✗
localStorage	✗	✗	✗
ActiveXObject	✓	✗	✗
indexedDB	✗	✗	✗
devicePixelRatio	✗	✗	✗
innerWidth	✓	✗	✗
innerHeight	✓	✗	✗
offsetHeight	✓	✓	✓
offsetWidth	✓	✓	✓

6.1.3. RandomSplit Results

The last Section focused on the EqualSplit strategy, which ensures that always the same web sites are used for training and testing. While this strategy is great for comparing different classifiers, it's also not completely representative for the real world. The problem here is that the training set and test set can contain certain peculiarities that are only present in those sets. Therefore, tackling this problem is important. For this purpose, a new strategy called *RandomSplit* is introduced. This strategy splits the tracking web sites and not-tracking web sites into two equally sized parts. This time, the web sites belonging to the training and test sets are chosen randomly. Therefore, the resulting sets are different for each execution. Please note that the nature of this way of splitting makes it not suitable to compare different classifiers, since in each execution, different web sites are used for training and testing. This new way of creating a training and test set is used to implement a further test for each classifier. The test is performed as follows:

1. Split the training data into a training and a test set using the RandomSplit function.
2. Train the classifier with the training set.
3. Classify each web site in the test set with the trained classifier.
4. Mesmerize the False-Positives, False-Negatives and correct ones.
5. Repeat step one to four **100** times.

The above mentioned test is executed for each classifier. The aggregated results were used to calculate the average values for False-Positives, False-Negatives and correct ones. Moreover, the standard deviation for each of the possible outcomes was computed. The standard deviation is considered to be the average scatter around the mean values.

Table 6.4 demonstrates that the test results for the EqualSplit are neither extremely good nor extraordinary bad. Overall, they are very similar to the results of the EqualSplit. This is very good news, because this means that the results summarized in Section 6.1.2 are sound and they do represent the whole training data. Moreover, the standard deviations show that the dispersion of the values is within the scope. In fact, the highest deviation

Table 6.4.: Summary of the performed tests using RandomSplit.

	Test	False-Positives		False-Negatives		Correct	
		Avg	σ	Avg	σ	Avg	σ
Aggregator	With Canvas	0.34	0.71	0.43	0.81	41.23	0.93
	Without Canvas	2.57	1.37	1.87	1.25	37.56	1.37
Bayes	With Canvas	1.23	1.57	0.46	0.85	40.31	1.43
	Without Canvas	5.58	2.42	0.74	1.14	35.68	2.30
Decision Tree	With Canvas	0.21	0.77	0.66	0.88	41.13	1.04
	Without Canvas	3.32	1.91	2.96	2.02	35.72	2.30
ExtraTrees	With Canvas	0.33	0.67	0.79	0.87	40.88	0.89
	Without Canvas	2.26	1.45	2.2	1.55	37.54	1.64

is 2.42 for the False-Positives of the Bayes classifier. Something that has to be highlighted is the fact that the number of correctly classified web sites is almost always very close to the expected value with RandomSplit. This is actually the most interesting number, because it directly reflects to the overall success rate of the classifier. Nevertheless, there are also some interesting differences concerning the False-Positives and False-Negatives. For example, the ExtraTrees Without Canvas test using EqualSplit (zero False-Positives, five False-Negatives, 37 correct) differs from the same test with RandomSplit. While the number of correct web sites is similar to the expected value (37 compared to 37.54 expected correct ones), the number for False-Positives is too low (zero compared to 2.26 expected False-Positives) and the number of False-Negatives is too high (five compared to 2.2 expected False-Negatives). Similar observations can be made when examining the Aggregator Without Canvas test results. Examining the standard deviations for the classifiers, one can assess that, again, the Aggregator outperforms the other classifiers. Among the basic classifiers - Naïve Bayes, Decision Tree and ExtraTrees - the standard deviations for the ExtraTrees were the lowest (1.45 False-Positives, 1.55 False-Negatives, 1.64 correct), with the exception for the False-Negatives for Naïve Bayes (only 0.74).

A further possibility to analyze the results of the RandomSplit tests are histograms. Those diagrams show the number of tests resulting in a specific number of False-Positive or False-Negative. The y-axis defines the number of tests and the x-axis denotes the total number of False-Positives or False-Negatives. In the following, the ExtraTrees test will be examined using histograms for False-Positives or False-Negatives. Since the analysis is the same for the other ones, only one classifier will be analyzed this way.

Figure 6.1 shows the histogram for the ExtraTrees test using RandomSplit and considering all features.

As one can see, about 80 tests yielded zero False-Positives and about 50 tests resulted in zero False-Negatives. Interestingly, the False-Negatives are distributed more evenly than the False-Positives, meaning that about 25 tests each yielded one or two False-Negatives. The False-Positives on the other hand only occur in about 25 tests. Summarizing, one can say that ExtraTrees tend to have a higher rate of False-Negatives. This is in line with the results shown in Table 6.4.

Figure 6.2 shows the histogram for the outcome of the test that does not consider Canvas-Fingerprinting-related features. As usual, there are a lot more misclassified web sites which can be seen in this Figure. Obviously, there are relatively few tests that did not yield any False-Positives or False-Negatives. Only in about seven tests, there were no False-Positives and only about 13 tests did not yield any False-Negatives. Most tests resulted in one or two False-Positives or False-Negatives. This outcome represents almost half of the tests. Approximately 32 tests led to three or four False-Positives and in about 25 cases, there were the same amount

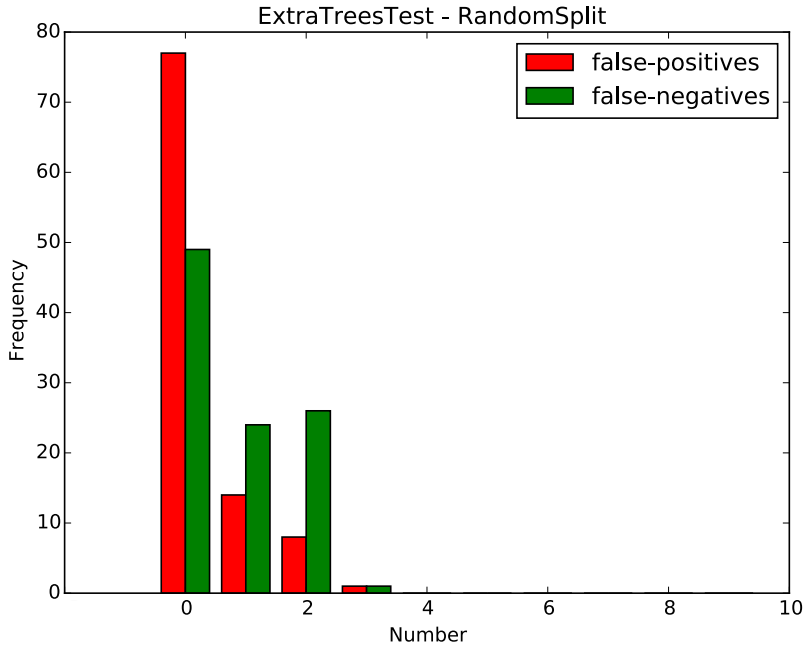

Figure 6.1.: Histogram for ExtraTrees with RandomSplit and all features.

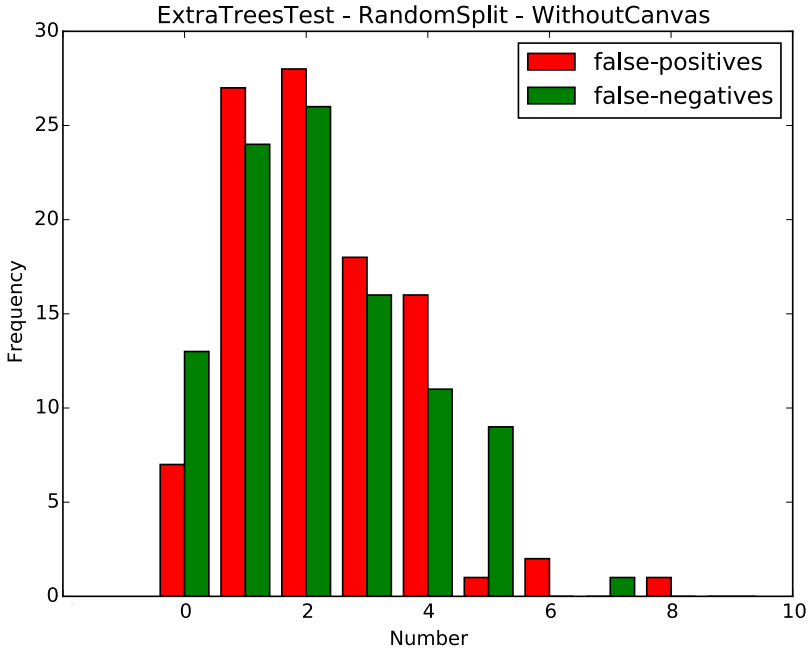

Figure 6.2.: Histogram for ExtraTrees with RandomSplit without Canvas-Fingerprinting-related features.

of False-Negatives. Surprisingly, almost ten tests yielded five False-Negatives.

To sum it up, it can be stated that almost every classifier is suitable to be used in such a detection system. However, removing the Canvas-Fingerprinting-related features directly influences the success rate significantly. Nevertheless, the success rates are - with the exception of Naïve Bayes - still very good, ranging from 86% to 88%.

6.2. Room for Improvement

This Section will show possible improvements to the detection system. Section 6.2.1 will focus on the training data and will show that there is a lot of hidden potential. Section 6.2.2 will illustrate that the classifiers themselves leave room for improvements, for example by tweaking the input parameters. Section 6.2.3 will state that another possible enhancement is to implement completely new classifiers. Last but not least, Section 6.2.4 will elucidate that also the sensors provide potential for improvement, for example by implementing new sensors or by improving the existing ones.

6.2.1. Training Data

The first approach to improve the detection rate is the set of training data. As it became clear, the classifiers heavily rely on it and all predictions are based on it. Consequently, the detection rate is directly linked to the quality of the training data. Right now, the set of training data encompasses 82 different web sites, with 41 being marked as tracking and 41 being marked as not-tracking. Of course, one possibility is to add further web sites to the set. While gathering new web sites that are not-tracking is a rather simple task, gathering web sites being considered to track their users is relatively complex. However, there are possibilities to get new tracking sites.

One option is to use the Chameleon browser extensions that has been presented in Section 3.2.1. This extension is really good when it comes to logging the site's features. There is even a tool called *Chameleon Crawler*[2] that offers browser automation with the Chameleon extension. This means that one can specify a list of URLs, which is than visited one after another and each URL is stored along with its features. After that, the summary of all crawled web sites and their related features can be seen in the web browser. Nevertheless, the results produced by Chameleon Crawler have to be examined manually and one must decide for each URL, if it is tracking the users or not. Chameleon does not offer a detection mechanism like the system presented in this thesis.

Another option to extend the training data is to use the system developed here. Section 5.1.3 showed, how the existing set of training data has been created. Now that the system can be used to detect fingerprinting attempts, one could just crawl URLs and use the system to predict the site's class. After that, the site and the related features are added to the data collection layer and are from now an part of the training data. However, this approach is a risky one, because, as shown in Section 6.1, the best classifier tested so far has a success rate of 88% when not all features are relevant. There is a small chance left that web sites are misclassified and consequently worsen the set of training data.

6.2.2. Optimizing Classifiers

Another point to consider concerns the current implementation of the classifiers. The prototype developed in this thesis relies on scikit-learn. However, due to the time constraints, it was not possible to optimize the classifiers by any means. This specifically concerns parameters. Every classifier of scikit-learn offers a vast range of parameters that can all be adjusted in order to enhance detection rates. One example is the splitting strategy for tree-based classifiers. The tests applied the Gini Index, but it is also imaginable to use Information Gain and see whether the results are better. Especially the ExtraTrees being one of the most complex classifiers tested here offers several other properties that can be optimized in order to improve the success rate.

For possible parameters of the classifiers, the reader is advised to visit the scikit-learn documentation [65, 66, 32].

6.2.3. Implementing New Classifiers

The three detection mechanisms implemented in this thesis are just a small selection of all the possibilities of Data Mining and Knowledge Discovery in Databases. It is possible that other detection mechanisms like

[2] `https://github.com/ghostwords/chameleon-crawler`

Support Vector Machines, Random Forest or Nearest Neighbors (just to mention a few) yield better results and are more suitable for this problem. Again, because of the limited scope of this work, the performance of those classifiers could not be evaluated. However, the prototype implementation's focus was on extensibility. Consequently, it is rather simple to extend the prototype and implement new classifiers from the scikit-learn library.

6.2.4. Improve Sensors

The system developed in this thesis does only consider one type of features: JavaScript-based ones. As it has already been mentioned, there are other technologies used for fingerprinting techniques like Java or Flash. In order to improve the detection rate of the classifiers, adding new sensors and new features and consequently extending the information used for training and classifying is a promising approach.

Another idea in this context is to reevaluate the list of features that should be considered. It has already been stated that a lot of features turned out to be not very telling concerning the classification of web site. This is something that has to be assessed. For example, features like `navigator.userAgent` are not suitable, because the feature's appearance on a web site does say nothing about its classification. As seen in Table 6.1, this feature is used on 41 of the tracking sites and 37 of the not-tracking sites.

Moreover, since the tree-based classifiers provide the possibility to print out the feature importances, this can be used to identify features, whose appearance on a web site is a strong hint towards its class. At best, there were four features taken into account (see Table 5.7). The feature list presented in 4.1 involves 27 features at the moment. It's quite obvious that there are a lot of features that are not applicable at all.

6.3. Summary

This Chapter discussed the measurements produced by the prototype implementation of the system. It became clear that the system is capable of detecting the usage of fingerprinting techniques. Section 6.1.2 showed that Canvas-Fingerprinting is almost always detectable with the Aggregator and Decision Tree classifiers yielding a success rate of 100%. However, the more general Browser-Fingerprinting is harder to detect, but the best classifier sill managed to correctly classify it with a success rate of 88%. Furthermore, it has been proofed that the results from Section 6.1.2 are not specific to the data used for training and classifying, but represent the complete the entire data set. Moreover, several possibilities to improve the detection rates of the classifiers have been proposed.

With a success rate of 100% for Canvas-Fingerprinting, it is safe to say that the developed system can effectively be utilized to detect this kind of fingerprinting technique. Although the more general Browser-Fingerprinting turned out the be harder to detect, the best classifier sill managed to correctly classify it with a success rate of 88%. The test results presented in Section 6.1.2 and their generalization is reflected on the daily use of the system. Web sites that are known to fingerprint their users are reliably detected. Consequently, new web sites not being part of the training set can also be classified with the system. This greatly enhances perceived privacy, because it is now possible to determine if the web site currently being visited is considered to track or not.

The number of False-Positives and False-Negatives is also something that needs to be reevaluated. As Section 6.2 illustrated in-depth, there are several possibilities to improve the system's detection rate. Even if it is not possible to eliminate all misclassified web sites, it might still be feasible to either decrease the number of False-Positives or False-Negatives. The decision, which one of those should be lowered depends on the purpose. If one aims to increase privacy as much as possible, the focus should be on a very low number of False-Negatives, because misclassifying a web site which is tracking, is worse than saying that a web site is tracking, which is in fact not. However, if privacy is not the focus, one could aim to lower the False-Positives and consequently not disturb or unnecessarily unsettle the user.

7. Conclusion and Future Work

Web-Tracking is omnipresent in the Internet. Using it enables trackers to collect huge amounts of personal data. All this happens in the background and unnoticed by the users. The growing number of individual devices each person is using leads to even more personal data being distributed on the Internet. This data is incredible valuable to a complete industry sector specializing in the field of gathering personal data and being worth multi billion dollars. Especially the advertisement sector is incredible keen on personal data, because those enable them to improve their advertisements and handcraft them for each user. The better the advertisements, the higher the chance that users click on them and consequently the higher the revenue of the carriers. Moreover, Web-Tracking is slowly becoming a thread to privacy, because with the growing amount of personal data, trackers are able to build up user profiles, representing the user's habits, affections and interests.

This thesis presented a system that tried to utilize classic detection mechanisms from the fields of Data-Mining and Knowledge Discovery in Databases to detect the usage of Web-Tracking on a web site. The system specialized in fingerprinting techniques involving Browser- and Canvas-Fingerprinting. Moreover, the system concentrated on only one technology used for realizing the fingerprinting techniques: JavaScript.

Using fingerprinting techniques enables the trackers to generate a unique fingerprint for each machine, making it possible to recognize users. This enables possible trackers to build profiles of the users, which constitutes a severe intrusion in the user's privacy. Another critical aspect of fingerprinting techniques is the fact that they are very hard to hamper or even to block.

The goal of the system developed in this thesis was to detect the usage of fingerprinting techniques on the fly. For this purpose, a general architectural design for a system has been proposed. The architecture consists of three different layers, with every layer specializing in one specific task. Chapter 4 demonstrated the basic concept behind the system.

As a proof of concept, the proposed system has been implemented. Four different classifiers are included: Naïve Bayes, Decision Tree, ExtraTrees and an Aggregator (combination of the other three classifiers). In order be able to test the performance of those, it was necessary to obtain a set of training data involving web sites that are considered to track and web sites, which are believed to not track their users.

The results of the classifiers were pretty good. In special, detecting Canvas-Fingerprinting is something, almost every classifier excels in. When it comes to detecting Browser-Fingerprinting, the success rates slightly dropped. Nevertheless, the best classifier had still a success rate of 88%. Summing it up, it is safe to say that the usage of fingerprinting techniques can be detected using traditional detection mechanisms.

While the system turned out to be able to detect fingerprinting techniques, this is just the beginning. The developed system can be improved in several aspects. For example, the system does not consider font enumeration at the moment, which is, as already mentioned, one of the strongest features. Implementing this could greatly enhance the detection rate and improving this rate is a general goal for future work. This thesis could only provide a basic proof of concept implementation. The system is - at the moment - not ready to be used in a production environment. But this would in fact be one of the most exciting things to do. Right now, only a small sample of the real world could be tested. By deploying the system into the wild, it could be evaluated, how the system performs in real life. Having a user base that utilizes the system actively and sends back feedback could greatly improve the whole system.

Reliably detecting is just one step in the right direction. The obvious next step would be hampering fingerprinting techniques. However, implementing counter-measures is a very difficult task. On the one hand, one has to ensure that the counter-measures do not result in the exact opposite: to even increase the fingerprintable surface of a machine. On the other hand, the system as it is designed right now is not capable of realizing certain counter-measures to fingerprinting techniques. The reason for this is the fact that the system's detection is based on the features that were accessed by a web site. However, once all those features are collected and

the web site is classified, it is already too late to implement counter-measures. So consequently, it would be necessary to redesign, if one future goal will be hampering fingerprinting techniques.

Finally, the developed detection system can be seen as first step towards a system that is capable of effectively detecting the use of fingerprinting techniques on web sites. Further developments in this area can lead to an increased privacy, which would be huge gain for every user that wants to protect his privacy when surfing in the Internet.

A. Training data

This Chapter contains the content of the training data.

A.1. Training set

This Section presents the training set, which have been used for training the classifiers.

Table A.1.: Training set yield by EqualSplit.

ID	URL	isTracking
1	http://ponpare.jp/	1
2	http://www.todayifoundout.com/	1
3	http://www.styletv.com.cn/	1
4	http://english.ctrip.com/	1
5	http://www.expedia.co.uk/	1
6	http://www.webroot.com/us/en/	1
7	http://www.ustream.tv/	1
8	https://www.foodpanda.in/	1
9	http://www.mvideo.ru/	1
10	http://www.lazada.co.id/	1
11	http://www.sears.ca/	1
12	http://www.sciencemag.org/	1
13	http://v5.ele.me/	1
14	http://fantasti.cc/	1
15	http://www.groupon.co.uk/	1
16	https://www.groupon.com/	1
17	http://www.groupon.com.br/	1
18	http://www.groupon.it/	1
19	http://guff.com/	1
20	http://www.hjenglish.com/	1
21	https://www.facebook.com/?_rdr	0
22	http://baidu.com/	0
23	http://www.baidu.com/	0
24	https://de.yahoo.com/?p=us	0
25	http://www.wikipedia.org/	0
26	http://www.amazon.com/	0
27	https://twitter.com/	0
28	http://www.taobao.com/market/global/index_new.php	0
29	http://www.qq.com/	0
30	https://www.linkedin.com/nhome/	0

31	https://login.live.com/login.srf?wa=wsignin1.0&rpsnv=12&ct=1433184657&rver=6.4.6456.0&wp=MBI_SSL_SHARED&wreply=https:%2F%2Fmail.live.com%2Fdefault.aspx%3Fshowunauth%3D1%26rru%3Dinbox&lc=1031&id=64855&mkt=de-DE&cbcxt=mai	0
32	http://www.sina.com.cn/	0
33	http://overseas.weibo.com/	0
34	http://www.yahoo.co.jp/	0
35	http://www.tmall.com/	0
36	http://www.ebay.com/	0
37	https://accounts.google.com/ServiceLogin?service=blogger&passive=1209600&continue=https://www.blogger.com/home?bpli%3D1&followup=https://www.blogger.com/home?bpli%3D1&ltmpl=start	0
38	http://www.hao123.com/	0
39	http://www.reddit.com/	0
40	http://www.bing.com/	0

A.2. Test set

This Section presents the test set - a list of web sites that were used for testing the classifiers.

Table A.2.: Test Set yield by EqualSplit.

ID	URL	isTracking
1	http://hypem.com/	1
2	http://video.tt/	1
3	http://www.washingtonpost.com/	1
4	https://vid.me/	1
5	http://www.linio.com.mx/	1
6	http://letitbit.net/	1
7	http://www.boulanger.com/	1
8	https://www.zalando.pl/	1
9	http://www.theaustralian.com.au/?nk=13606d45f20ce9b84fa5dff357ee1ed4	1
10	http://pikabu.ru/	1
11	http://www.lamoda.ru/	1
12	https://robertsspaceindustries.com/	1
13	http://dantri.com.vn/	1
14	http://us.webnode.com/	1
15	http://www.webnode.cz/	1
16	http://www.inquisitr.com/	1
17	http://www.philly.com/	1
18	http://www.eater.com/	1
19	http://www.salon.com/	1
20	http://www.cheatsheet.com/	1
21	http://www.pof.de/	1
22	http://www.sohu.com/	0
23	http://www.amazon.co.jp/	0
24	https://www.tumblr.com/	0
25	http://t.co/	0
26	http://imgur.com/	0
27	https://de.wordpress.com/	0
28	https://www.pinterest.com/	0
29	http://instagram.com/	0
30	http://www.msn.com/de-de/	0
31	https://www.paypal.com/de/webapps/mpp/home	0
32	http://www.apple.com/	0
33	http://www.microsoft.com/de-de/default.aspx	0
34	https://www.google.it/?gws_rd=ssl	0
35	http://www.imdb.com/	0
36	http://fc2.com/	0
37	http://www.xvideos.com/	0
38	http://www.aliexpress.com/	0
39	http://www.alibaba.com/	0
40	http://stackoverflow.com/	0
41	http://360.cn/	0
42	http://de.ask.com/?o=312&l=dir	0

A.3. Complete training data

This Section presents the complete set of training data.

Table A.3.: Overview of all web sites of the training data.

ID	URL	isTracking
1	http://ponpare.jp/	1
2	http://www.todayifoundout.com/	1
3	http://www.styletv.com.cn/	1
4	http://english.ctrip.com/	1
5	http://www.expedia.co.uk/	1
6	http://www.webroot.com/us/en/	1
7	http://www.ustream.tv/	1
8	https://www.foodpanda.in/	1
9	http://www.mvideo.ru/	1
10	http://www.lazada.co.id/	1
11	http://www.sears.ca/	1
12	http://www.sciencemag.org/	1
13	http://v5.ele.me/	1
14	http://fantasti.cc/	1
15	http://www.groupon.co.uk/	1
16	https://www.groupon.com/	1
17	http://www.groupon.com.br/	1
18	http://www.groupon.it/	1
19	http://guff.com/	1
20	http://www.hjenglish.com/	1
21	http://hypem.com/	1
22	http://video.tt/	1
23	http://www.washingtonpost.com/	1
24	https://vid.me/	1
25	http://www.linio.com.mx/	1
26	http://letitbit.net/	1
27	http://www.boulanger.com/	1
28	https://www.zalando.pl/	1
29	http://www.theaustralian.com.au/?nk=13606d45f20ce9b84fa5dff357ee1ed4	1
30	http://pikabu.ru/	1
31	http://www.lamoda.ru/	1
32	https://robertsspaceindustries.com/	1
33	http://dantri.com.vn/	1
34	http://us.webnode.com/	1
35	http://www.webnode.cz/	1
36	http://www.inquisitr.com/	1
37	http://www.philly.com/	1
38	http://www.eater.com/	1
39	http://www.salon.com/	1
40	http://www.cheatsheet.com/	1
41	http://www.pof.de/	1
42	https://www.facebook.com/?_rdr	0
43	http://baidu.com/	0
44	http://www.baidu.com/	0
45	https://de.yahoo.com/?p=us	0

46	http://www.wikipedia.org/	0
47	http://www.amazon.com/	0
48	https://twitter.com/	0
49	http://www.taobao.com/market/global/index_new.php	0
50	http://www.qq.com/	0
51	https://www.linkedin.com/nhome/	0
52	https://login.live.com/login.srf?wa=wsignin1.0&rpsnv=12&ct=1433184657&rver=6.4.6456.0&wp=MBI_SSL_SHARED&wreply=https:%2F%2Fmail.live.com%2Fdefault.aspx%3Fshowunauth%3D1%26rru%3Dinbox&lc=1031&id=64855&mkt=de-DE&cbcxt=mai	0
53	http://www.sina.com.cn/	0
54	http://overseas.weibo.com/	0
55	http://www.yahoo.co.jp/	0
56	http://www.tmall.com/	0
57	http://www.ebay.com/	0
58	https://accounts.google.com/ServiceLogin?service=blogger&passive=1209600&continue=https://www.blogger.com/home?bpli%3D1&followup=https://www.blogger.com/home?bpli%3D1&ltmpl=start	0
59	http://www.hao123.com/	0
60	http://www.reddit.com/	0
61	http://www.bing.com/	0
62	http://www.sohu.com/	0
63	http://www.amazon.co.jp/	0
64	https://www.tumblr.com/	0
65	http://t.co/	0
66	http://imgur.com/	0
67	https://de.wordpress.com/	0
68	https://www.pinterest.com/	0
69	http://instagram.com/	0
70	http://www.msn.com/de-de/	0
71	https://www.paypal.com/de/webapps/mpp/home	0
72	http://www.apple.com/	0
73	http://www.microsoft.com/de-de/default.aspx	0
74	https://www.google.it/?gws_rd=ssl	0
75	http://www.imdb.com/	0
76	http://fc2.com/	0
77	http://www.xvideos.com/	0
78	http://www.aliexpress.com/	0
79	http://www.alibaba.com/	0
80	http://stackoverflow.com/	0
81	http://360.cn/	0
82	http://de.ask.com/?o=312&l=dir	0

B. Content of CD

The attached CD contains the following additional information:

1. *Sources*
   - *firevest*
     Prototype implementation of a Firefox extension, that used the object poisoning technique presented in Section 5.1.1 to log the accessed features of a web site.
   - *firehelmet*
     Prototype implementation of a Firefox extension utilizing the static analysis approach shown in Section 5.1.1. Please note, that this extension is not working correctly, since it's not capable of logging all accessed features.
   - *fireshorts*
     Backend implementation of a web server. This software is responsible for the data collection and logic/evaluation.
2. *Database*
   This directory contains a database dump of the training data. The dump is suited for a MySQL Server v5.6.21.

C. Installation

This Chapter will explain, how the developed system can be installed, in order to be able to recreate the results. Section C.1 will show, how the database is set up. After that, Section C.2 will show the installation of the Firefox extension called *Firevest*. Lastly, Section C.3 will provide information about, how the server side, *Fireshorts* can be installed.

C.1. Database setup

The CD contains a dump file of the MySQL data used for training and testing. The database utilized for the development, has been a MySQL Community Server v5.6.21. The dump was created using *mysqldump*[1].

Restoring the database is plain simple. At first, one must create a database with a suitable name. After that, the database can be restored using the dump file. Listing C.1 shows the basic command. Please note, that this assumes, that the MySQL Server is running on the *localhost* on its default port 3306.

Listing C.1: Restoring the database.

```
1 mysql -u <user> -p<password> <name_of_database> < firevest.sql
```

After restoring the database, it can be used by the data collection and logic/evaluation layers.

C.2. Firevest

The Firefox extension has been developed using the Addon-SDK provided by Mozilla [67]. This SDK helps the user to create new Firefox extensions by offering numerous APIs, that facilitate the daily work. In special, the most modern version of the SDK has been used: *jpm* [68]. jpm is distributed with the Node.js package manager (npm). However, this tool can only be used with Firefox v38 onwards. It eases the development of a browser extensions by providing several helpers to, for example, initiate a new project, run the current project using a Firefox instance and a lot more.

To run *Firevest*, one has to install *jpm* using *npm*. Obviously, Firefox has to be installed as well. After that, one can switch to the firevest directory in a terminal and run `jpm run`. After that, a Firefox instance opens with the current version of the Firevest extension being installed. Figure C.1 shows the current available settings of the extensions. Right now, there is only one setting changeable. This concerns the interface, the collected features should be sent to. Activating this settings leads to the extensions sending the features to the `/collect` interface of the server, that is used for building training data. Otherwise, the `/classify` interface is used, which - as the name suggests - classifies the input and sends back True (1) or False (0).

C.3. Fireshorts

This project represents the server-side of the whole detection system. Consequently, it contains the data collection and logic/evaluation layers.

The Fireshorts project does have the following dependencies, which can be installed via `pip(3)`.

[1] `https://dev.mysql.com/doc/refman/5.1/en/mysqldump.html`

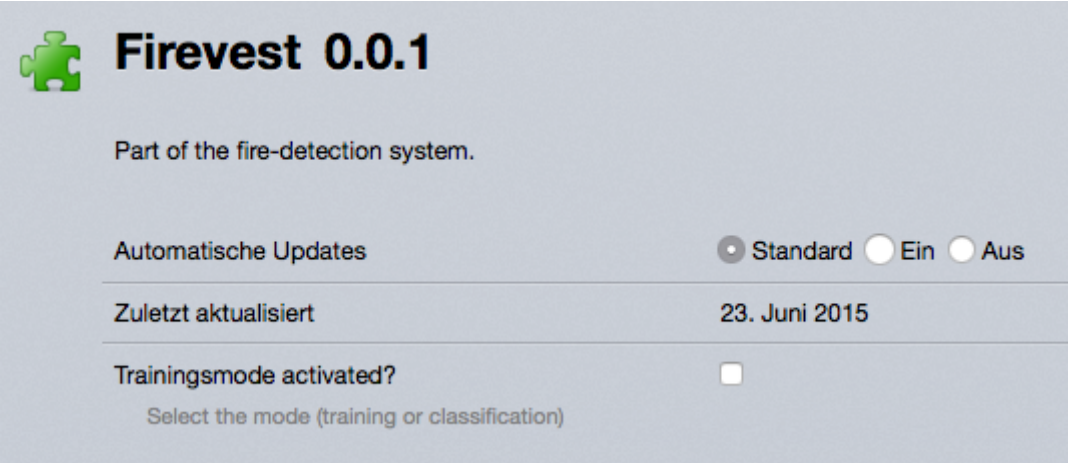

Figure C.1.: Firevest settings.

- peewee: Object-relational mapping framework
- flask: web framework
- PyMySQL: MySQL library

Fireshorts is a web server based on *flask*. Running it is extremely simple. Just start the *Fireshorts.py* by calling `python3 Fireshorts.py`.

The following list will provide basic information about each package/file:

1. *fireshorts*
   - *classifier*
     This package contains all implemented classifiers. In *BaseClassifier.py*, their superclass located. This class implements most of the classifiers functionality.
   - *db*
     Here, the DAO is implemented. This concerns the storage of new web sites and their features.
   - *model*
     This package involves the relevant models. Those have already been depicted in Figure 5.2. As mentioned before, *peewee* is used as ORM-framework.
   - *util*
     This package contains several utility modules. Most notably, the Settings.py file, that involves settings like the classifier to use for classification.
2. *templates*
   In this package, template-files are stored. This is used to either serve dynamic HTMl content or to write the relevant LaTeX-Tables.
3. *test*
   Package, that contains the tests for each classifier.
4. *Fireshorts.py*
   Main file of the system. Here, interfaces are defined and the main logic of the system is implemented.

Bibliography

[1] Whatsapp sails past sms, but where does messaging go next?, 1 2015. URL http://benevans.com/benedictevans/2015/1/11/whatsapp-sails-past-sms-but-where-does-messaging-go-next.

[2] Edward snowden: Prism, June 2013. URL http://www.washingtonpost.com/investigations/us-intelligence-mining-data-from-nine-us-internet-companies-in-broad-secret-program/2013/06/06/3a0c0da8-cebf-11e2-8845-d970ccb04497_story.html.

[3] Edward snowden: Tempora, June 2013. URL http://www.thewire.com/national/2013/06/uk-tempora-program/66490/.

[4] Referer header. URL http://en.wikipedia.org/wiki/HTTP_referer.

[5] Balachander Krishnamurthy, Konstantin Naryshkin, and Craig E. Wills. Privacy leakage vs. protection measures: the growing disconnect. *Web 2.0 Security and Privacy Workshop*, May 2011.

[6] Sites feed personal details to new tracking industry, July 2010. URL http://www.wsj.com/articles/SB10001424052748703977004575393173432219064.

[7] Markus Schneider, Matthias Enzmann, and Martin Stopczynski. Web-tracking-report 2014. Technical Report SIT-TR-2014-01, Fraunhofer-Institut für Sichere Informationstechnologie, February 2014.

[8] PWC and Interactive Advertising Bureau. Internet advertising revenue full-year report 2014, July 2015.

[9] Ein bild aus tausend spuren, August 2013. URL http://www.zeit.de/2013/32/datenspuren-internet-snowden-prism-tempora.

[10] Giving the web a memory cost its users privacy. URL http://nytimes.com/2001/09/04/technology/04COOK.html.

[11] Peter Eckersley. How unique is your web browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, pages 1–18, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-14526-4, 978-3-642-14526-1. URL http://dl.acm.org/citation.cfm?id=1881151.1881152.

[12] Mozilla wiki: Fingerprinting. URL https://wiki.mozilla.org/Fingerprinting.

[13] Fingerprint.js, . URL https://github.com/Valve/fingerprintjs.

[14] Fingerprint.js2, . URL https://github.com/Valve/fingerprintjs2.

[15] Javascript navigator object, . URL https://developer.mozilla.org/de/docs/Web/API/Navigator.

[16] Javascript screen object, . URL https://developer.mozilla.org/en-US/docs/Web/API/Screen.

[17] Murmurhash. URL http://en.wikipedia.org/wiki/MurmurHash.

[18] Html5 specification, . URL http://www.w3.org/TR/html5/.

[19] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In Matt Fredrikson, editor, *Proceedings of W2SP 2012*. IEEE Computer Society, May 2012.

[20] Canvas in html5, . URL http://www.w3schools.com/tags/ref_canvas.asp.

[21] Font detection using js and css. URL http://www.lalit.org/lab/javascript-css-font-detect/.

[22] Webgl specification. URL https://www.khronos.org/registry/webgl/specs/1.0/.

[23] How big data has changed healthcare, . URL http://www.investopedia.com/articles/investing/042815/how-big-data-has-changed-healthcare.asp.

[24] Go big or go home: How to utilize big data for human resources, . URL http://www.business.com/human-resources/how-to-utilize-big-data-for-human-resources/.

[25] Big data: The winning formula in sports, . URL http://www.forbes.com/sites/bernardmarr/2015/03/25/big-data-the-winning-formula-in-sports/.

[26] Without good analysis, big data is just a big trash dump, . URL http://www.entrepreneur.com/article/246470.

[27] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining Concepts and Techniques*. Morgen Kaufmann, 2011.

[28] Thomas bayes. URL http://en.wikipedia.org/wiki/Thomas_Bayes.

[29] Paul graham - a plan for spam, August 2002. URL http://www.paulgraham.com/spam.html.

[30] Thunderbird. URL http://en.wikipedia.org/wiki/Mozilla_Thunderbird.

[31] Knowledge discovery in databases i - ss2014. URL http://www.dbs.ifi.lmu.de/cms/Knowledge_Discovery_in_Databases_I_%28KDD_I%29_14.

[32] Scikit-learn: Decision tree, . URL http://scikit-learn.org/stable/modules/tree.html.

[33] PD Dr. Arthur Zimek and Dr. Erich Schubert. Knowledge discovery in databases i. Course, 2014.

[34] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63 (1):3–42, 2006. ISSN 0885-6125. doi: 10.1007/s10994-006-6226-1. URL http://dx.doi.org/10.1007/s10994-006-6226-1.

[35] Keaton Mowery, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. Fingerprinting information in JavaScript implementations. In Helen Wang, editor, *Proceedings of W2SP 2011*. IEEE Computer Society, May 2011.

[36] Jonathan R. Mayer. *"Any person... a pamphleteer": Internet Anonymity in the Age of Web 2.0*. PhD thesis, Princeton University, 2009.

[37] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 541–555, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-0-7695-4977-4. doi: 10.1109/SP.2013.43. URL http://dx.doi.org/10.1109/SP.2013.43.

[38] Jonathan R. Mayer and John C. Mitchell. Third-party web tracking: Policy and technology. In *IEEE Symposium on Security and Privacy*, pages 413–427. IEEE Computer Society, 2012. ISBN 978-0-7695-4681-0. URL http://dblp.uni-trier.de/db/conf/sp/sp2012.html#MayerM12.

[39] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. Fpdetective: Dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 1129–1140, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2477-9. doi: 10.1145/2508859.2516674. URL http://doi.acm.org/10.1145/2508859.2516674.

[40] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 674–689, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2957-6. doi: 10.1145/2660267.2660347. URL http://doi.acm.org/10.1145/2660267.2660347.

[41] Selenium. URL http://www.seleniumhq.org/.

[42] Paul Stone. Pixel perfect timing attacks with html5. Technical report, context Information Security, July 2013.

[43] Robert Kotcher, Yutong Pei, Pranjal Jumde, and Collin Jackson. Cross-origin pixel stealing: timing attacks using css filters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, CCS '13, pages 1055–1062, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2477-9. doi: 10.1145/2508859.2516712. URL http://doi.acm.org/10.1145/2508859.2516712.

[44] Addthis: Reaction on the canvas element usage, July 2014. URL http://www.addthis.com/blog/2014/07/23/the-facts-about-our-use-of-a-canvas-element-in-our-recent-rd-test/#.VViNJ5PtlBc.

[45] Chameleon extension. URL https://github.com/ghostwords/chameleon.

[46] Canvasblocker. URL https://addons.mozilla.org/de/firefox/addon/canvasblocker/.

[47] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. Privaricator: Deceiving fingerprinters with little white lies. Technical Report MSR-TR-2014-26, February 2014. URL http://research.microsoft.com/apps/pubs/default.aspx?id=209989.

[48] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. User tracking on the web via cross-browser fingerprinting. In *Proceedings of the 16th Nordic Conference on Information Security Technology for Applications*, NordSec'11, pages 31–46, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-29614-7. doi: 10.1007/978-3-642-29615-4_4. URL http://dx.doi.org/10.1007/978-3-642-29615-4_4.

[49] Tor browser. URL https://www.torproject.org/projects/torbrowser.html.en.

[50] Firegloves. URL http://fingerprint.pet-portal.eu/?menu=6.

[51] Ameya Nayak, Anil Poriya, and Dikshay Poojary. Type of nosql databases and its comparison with relational databases. *International Journal of Applied Information Systems*, 5(4):16–19, March 2013. Published by Foundation of Computer Science, New York, USA.

[52] Neal Leavitt. Will nosql databases live up to their promise? *Computer*, 43(2):12–14, February 2010. ISSN 0018-9162. doi: 10.1109/MC.2010.58. URL http://dx.doi.org/10.1109/MC.2010.58.

[53] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. Pearson, 5 edition, 2011.

[54] Jürgen Scherff. *Grundkurs Computernetzwerke*. Vieweg + Teubner, 2 edition, 2010.

[55] Prof. Dr. Claudia Eckert. *IT-Sicherheit*. Oldenbourg Verlag München, 8 edition, 2013.

[56] The cia triad. URL http://www.techrepublic.com/blog/it-security/the-cia-triad/.

[57] Using oauth 2.0 to access google apis. URL https://developers.google.com/identity/protocols/OAuth2.

[58] Ast specification of javascript, . URL https://github.com/estree/estree.

[59] Estraverse, . URL https://github.com/estools/estraverse.

[60] David Herman. *Effective JavaScript*. Addison-Wesley, 2013.

[61] Javascript object.defineproperty, . URL https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty.

[62] Javascript globaleventhandlers. URL https://developer.mozilla.org/de/docs/Web/API/GlobalEventHandlers/onload.

[63] The r project for statistical computing. URL http://www.r-project.org/about.html.

[64] Scitkit-learn: Machine learning in python, . URL http://scikit-learn.org/stable/.

[65] Scikit-learn: Gaussian naive bayes, . URL http://scikit-learn.org/0.15/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB.

[66] Scikit-learn: Extratrees, . URL http://scikit-learn.org/0.15/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html#sklearn.ensemble.ExtraTreesClassifier.

[67] Mozilla addon-sdk, . URL https://developer.mozilla.org/en/Add-ons/SDK.

[68] Mozilla addon-sdk: Jpm, . URL https://developer.mozilla.org/en-US/Add-ons/SDK/Tools/jpm.