Technische Universität Berlin

Faculty IV - Electrical Engineering and Computer Science
Chair for Security in Telecommunications

Ernst-Reuter-Platz 7
10587 Berlin
http://www.isti.tu-berlin.de/

Master Thesis

Uncovering Obfuscated Fingerprinting Techniques. A Large Scale Security and Privacy Analysis of Device Identification.

Felix Kybranz

Matriculation Number: 380341
kybranz@campus.tu-berlin.de

13.08.2020

Supervised by:
Prof. Dr. Jean-Pierre Seifert

Second Supervisor:
Prof. Dr. Florian Tschorsch

Assistant Supervisor:
M.Sc. Julian Fietkau

Abstract

The privacy and security of personal information is the basis for developing trust and confidence on the Internet. Companies have been protecting this information by, among other things, encrypting sensitive data in critical processes to ensure trust and a secure environment for users. In addition, they are transforming the Web into an increasingly personalized environment in order to differentiate individual users by the characteristics of their online behavior or digital fingerprints. That way, they are able to provide user-specific content and targeted advertisements. However, the technology used for taking fingerprints has access to sensitive user information and is capable of creating traceable identities of individuals without their awareness or explicit consent. Although fingerprinting can serve to limit abusive activities, such as credit card fraud, it also provides possibilities for abuse.

In this thesis, we show a way to evaluate the techniques used to create digital fingerprints, which so far has been prevented by typical defense mechanisms, such as code obfuscation in web browsers. For that, we created a browser extension that allows us to actively monitor the behavior of third-party JavaScript code in real-time. In addition, we review other privacy extensions and settings and discuss the impact and consequences of the lack of transparency on user privacy. In order to prove the applicability and precision of our techniques, we perform an analysis of fingerprint activity on Alexa's 10.000 most important websites and identify clusters with identical signatures of fingerprinting scripts on different websites. As a result, we can identify networks of websites that use the same or similar fingerprint scripts and draw conclusions as to whether websites are likely to be affiliated.

Zusammenfassung

Der Schutz und die Sicherheit personenbezogener Daten ist die Voraussetzung für die Entwicklung von Vertrauen und Sicherheit im Internet. Unternehmen haben diese Informationen unter anderem durch die Verschlüsselung sensibler Daten in kritischen Prozessen geschützt, um Vertrauen und eine sichere Umgebung für die Nutzer zu gewährleisten. Darüber hinaus transformieren sie das Web in eine zunehmend personalisierte Umgebung, sodass einzelne Benutzer anhand der Merkmale ihres Onlineverhaltens, bzw. ihres digitalen Fingerabdrucks, unterschieden werden können. Auf diese Weise sind sie in der Lage, nutzerspezifische Inhalte und gezielte Werbung anzubieten. Die für die Erstellung von Fingerabdrücken verwendete Technologie hat jedoch auch Zugriff auf sensible Nutzerinformationen und ist in der Lage, unterscheidbare Identitäten von Nutzern ohne deren Wissen oder ausdrückliche Zustimmung zu erstellen. Obwohl die Erfassung von Fingerabdrücken genutzt werden kann um missbräuchliche Aktivitäten, wie Kreditkartenbetrug, einzuschränken, eröffnet sie gleichzeitig Möglichkeiten zum Missbrauch.

In dieser Arbeit zeigen wir einen Weg auf, um Techniken zur Erstellung digitaler Fingerabdrücke zu bewerten, was bisher durch typische Mechanismen, wie z.B. Codeverschleierung in Webbrowsern, verhindert wurde. Aus diesem Grund haben wir eine Browsererweiterung entwickelt, die es uns erlaubt, das Verhalten von JavaScript Code Dritter in Echtzeit zu überwachen. Darüber hinaus testen wir vorhandene Browsererweiterungen und Einstellungen, die die Privatsphäre schützen sollen und diskutieren die globalen Auswirkungen und Folgen der mangelnden Transparenz auf die Privatsphäre der Nutzer. Um die Anwendbarkeit und Wirksamkeit unserer Techniken nachzuweisen, überwachen wir die Erstellung und die Extraktion von Fingerabdrücken durch Skripte Dritter auf den 10.000 wichtigsten Webseiten nach Alexa. Weiterhin identifizieren wir Cluster mit identischen Fingerabdrucksignaturen der verschiedenen Webseiten. Auf diese Weise können wir Netzwerke von Webseiten aufzeigen, welche die gleichen Herkunftsnetzwerke oder sogar identische Fingerabdruckskripte verwenden, und Schlussfolgerungen ziehen, ob diese möglicherweise miteinander in Verbindung stehen.

Declaration of Authorship

I, Felix KYBRANZ, hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

The independent and unaided completion of the thesis is affirmed by affidavit:

Signed: _____

Date: _____

Acknowledgements

First, I would like to acknowledge Einstein-Professor Dr. Jean-Pierre Seifert for giving me the opportunity to write my thesis in his team and for inspiring my interest in IT security. Further, I would like to thank my assistant supervisor Julian Fietkau for his guidance, his many valuable suggestions, and ideas for improvement throughout the thesis. I would also like to thank all other members of the SecT department for the very enjoyable working environment and for everything they have made possible for me.
My special thanks go to Ioannis Tourountzis, Peter Schüllermann, Philipp Nickel, Yuri Gbur, and all the others who have taught me a lot and contributed relentlessly to shape my way of thinking.
But most of all, I have to thank my family, who helped with everything and encouraged me for as long as I can remember, Irma and Alea for being the best cats in the world, and especially Sylvi Schmeiß for all her love and support.

Contents

Abbreviations

CAPTCHA	**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part
DOM	**D**ocument **O**bject **M**odel
CSSOM	**C**ascading **S**tyle **S**heet **O**bject **M**odel
JIT	**J**ust **I**n **T**ime compiler
OS	**O**peration **S**ystem
GDPR	**G**eneral **D**ata **P**rotection **R**egulation
CSS	**C**ascading **S**tyle **S**heets
2FA	Two(**2**) **F**actor **A**uthentication
REST	**RE**presentational **S**tate **T**ransfer
URL	**U**niform **R**esource **L**ocator
HTTPS	**H**yper**T**ext **T**ransfer **P**rotocol **S**ecure
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
CSV	**C**omma **S**eperated **V**alue
OSI	**O**pen **S**ystem **I**ntegration
IT	**I**nformation **T**echnology
WWW	**W**orld **W**ide **W**eb
UDP	**U**ser **D**ata **P**rotocol
TCP	**T**ransmission **T**ransfer **P**rotocol
IP	**I**nternet **P**rotocol address
NMAP	**N**etwork **MAP**per
MITM	**M**an **I**n **T**he **M**iddle
TTL	**T**ime **T**o **L**ive
API	**A**pplication **P**rogramming **I**nterface
SMS	**S**hort **M**essage **S**ervice
DNS	**D**omain **N**ame **S**ervice
DHCP	**D**ynamic **H**ost **C**onfiguration **P**rotocol
VM	**V**irtual **M**achine
FPMON	**F**inger**P**rinting **MON**itor
FPCRAWL	**F**inger**P**rinting **CRAWL**er
CPU	**C**entral **P**rocessing **U**nit
KVM	**K**ernel-based **V**irtual **M**achine
LVM	**L**ogical **V**olume **M**anager
IDE	**I**ntegrated **D**evelopment **E**nvironment
EU	**E**uropean **U**nion
RAM	**R**andom **A**ccess **M**emory
VPN	**V**irtual **P**rivate **N**etwork
CDN	**C**ontent **D**elivery **N**etwork
NGO	**N**on-**G**overment **O**rganization
PNG	**P**ortable **N**etwork **G**raphics

SMB	**S**erver **M**essage **B**lock
TLS	**T**ransport **L**ayer **S**ecurity
SSL	**S**ecure **S**ockets **L**ayer
SNMP	**S**imple **N**etwork **M**anagement **P**rotocol
NetBIOS	**Net**work **B**asic **I**nput **O**utput **P**rotocol
ICMP	**I**nternet **C**ontrol **M**essage **P**rotocol
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
CDP	**C**isco **D**iscovery **P**rotocol

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The Internet is a global network of interconnected computer hardware and software systems, making possible the storage, retrieval, circulation, and processing of information and communication across time and space [1]. It is shaping all of our lives by how we communicate with each other, how we exchange and process information, and how we explore new things. Regardless of where we live or who we are, we can immediately retrieve knowledge from digital encyclopedias like Wikipedia or share information on Social Media, like Reddit or Twitter [2, 3, 4]. Access to the Internet has become a fundamental element, whether in combination with our work or in our spare time.

However, such improvements not only add capabilities that can be used to improve the quality of life but also offer the potential for abuse. Since the early days of the Internet, there have always been individuals or organizations who give themselves an advantage by having a more in-depth knowledge of the underlying technologies than the average user or owners of online services. These players are more and more frequently discovering new ways to exploit users and other companies. In the vast majority of cases, this is done in a way unknown to the respective users, and in some of them in areas that have not yet been regulated [5].

One of the areas in which significant progress has been made in recent years is the area we will examine in this thesis: the fingerprinting of web devices. The challenges, however, that arise when using this technology are not directly visible. When taking fingerprints from web devices, sensitive user information is shared without the explicit consent and often even without the awareness of an affected user [5]. Although, protective regulations such as the General Data Protection Regulation (GDPR) specify precise rules for the collection of fingerprints in order to limit their abuse and to protect the security and privacy of user data but have difficulties in putting their rules into practice [6, 7]. To address issues like this, further extensive development in that field is needed to supervise potential privacy-violating activities transparently and ensure that the Internet remains unbiased and free as it should be.

1.2 Threat Model

This thesis is written according to the following assumptions, which reflect the current landscape of the Internet on the topic of web-device fingerprinting.

Users typically access the Internet through a web browser, which consists of an ecosystem of different technologies. This ecosystem offers a lot of utility and freedom and serves any purpose a user could need to interact with external Internet services, like websites. From media players, PDF readers, and source code compilers to APIs that enable interactions with native underlying system components, the browser behaves like an operating system itself, located in the application layer of the OSI model [8]. In the OSI reference model, the communications between a computing system are split into seven different abstraction layers: Physical, Data Link, Network, Transport, Session, Presentation, and Application [9]. When a user visits a website, code is transferred and executed on a client's machine through the browser. Since a web browser itself is an application, one would think that it performs all its actions exclusively at the application layer. The browser, however, has access to components and protocols much lower in the OSI model. That property of having extended privileges is one of the main elements that enables third-parties to perform fingerprinting actions, which we will examine in this thesis. Additionally, the transmitted code is often too complicated and complex for the average user to gain insight into its functionality. Moreover, code is often obfuscated, which makes it unreadable and, further, intentionally obscures its purpose. This thesis, therefore, focuses on examining the impact on security and privacy of obfuscated code execution when performing fingerprinting actions that extract and capture sensitive user information.

In recent years, browser capabilities have often been misused for a number of attacks, such as large-scale credit card fraud or the extraction of user information by dubious advertising companies all over the world [10]. Especially the Magecard attacks in 2019, as shown in [11], impressively introduced to the world the abilities and dangers of these features. Repositories of third-party JavaScript scripts were compromised and modified to deliver a slightly modified version instead of the original script to silently skim and extract information from credit card form elements on various payment pages on e-commerce sites. The fatal element of this attack is that the malicious code is automatically distributed to all websites that use the affected third-party code. It often remains undiscovered for days or months, since the code is part of the trusted asset of a website.

As many users and websites are affected by this intrusive type of action, there is an increasing need for transparency to ensure the security of user data processed in the online world. Further, conditions and precautions are required to strengthen the trust as well as the privacy of its users.

1.3 Goal of Thesis

The goal of this thesis is to investigate fingerprinting techniques that websites use on client web devices in order to discern the current status of this technology. To achieve this, we will develop a browser extension that actively monitors the behavior of websites despite preventive measurements such as code obfuscation. With this approach, we will gain insight into the fingerprinting activities, from a visiting user perspective, that would otherwise remain hidden. This strategy probably will not cover all fingerprinting techniques available today, but promises a way to detect the potential aggressiveness and the scope of fingerprinting abilities of a company, which will be used to assess the impact on the privacy and security of sensitive user data. Another objective is to further improve transparency and privacy awareness among Internet users by enabling them to assess potential risks themselves using this browser extension. In addition, we will use the tools we have developed to scan the Internet on a large scale to identify and analyze fingerprint mechanisms and to analyze the distribution of related networks based on the fingerprint signatures we measure. With this, we will create a way to identify links and relationships between websites and advertising networks using the same tracking mechanisms like they use to identify their users.

Parts of this thesis have also resulted in a publication that is currently under review.

1.4 Structure of Thesis

The thesis is divided into six chapters, each covering a fundamental part of this work. The very first Chapter 1 explains the motivation of this work and examines the threat model of the topic. Chapter 2 introduces the reader to all technologies and basic concepts and provides the background for a comprehensive understanding of the work we perform in the following chapters. In addition, we will mention the most important related work regarding the field of this thesis. In Chapter 3, we introduce the design of the solutions we develop and discuss the overall concept behind the architecture we use and conduct our analyses with. Chapter 4 discusses our implementation of the concept from the previous chapter. We will then explain in detail any components we have used and addressed the issues we have had problems with. Chapter 5 introduces our analysis and presents the results we have obtained in this thesis. These are discussed in greater detail and further elaborated together with various advanced fingerprinting techniques and their privacy implications. The last Chapter 6 summarizes and concludes our methods and results and gives a general overview of the most important tasks solved in this work. It further outlines open questions that could be part of future research.

Chapter 2

Background and Related Work

In this chapter, the reader will get an overview of the key components, terms, and technology basics necessary to understand the following work. With the aim to highlight and to point out differences between already existing approaches, this chapter will also showcase examples of how the industry is already applying variant fingerprinting techniques. Further, it will reference the most relevant related work form the past years and explain who else is working on this topic in this research area.

2.1 Fingerprinting Classes

A fingerprint is an impression left by the friction ridges of a human finger and is an important method of forensic science. The forensic value is provided by the properties of the fingerprint, which are stability and diversity. In an ideal world, fingerprints are consistent, which means that their attributes never change (stability), and fingerprints are uniquely identifiable through thousands and millions of other fingerprints (diversity). Since the early 1900, biometrical identification methods are the dominating approach to identify a person uniquely. Law enforcement and other organization around the world are using this method for decades now, and the motivation of having a digital equivalent, for example for signing documents online [12], is exceptionally high since the beginning and rise of the WWW in the late nineties.

For identification, IT systems offer a lot of possibilities and vectors for analyzing behavior on all layers of the OSI model [9]. Various protocols that process and exchange data have a somewhat unique approach doing their task and are, therefore, a target for fingerprinting. Different behavior is measured to distinguish between protocol or software used. Even across slightly different versions of the same software, small variations can be measured to differentiate the target software. In the following Table 2.1 well-established protocols like IPv4, FTP, or DHCP are shown, that were targets of fingerprint analysis in the past and are subject to current research.

Layer	Name	Protocol	Research
7	Application	SMB, HTTP, Telnet, TLS/SSL, DHCP	[13], [14], [15], [16], [17]
5	Session	SNMP, NetBIOS	[18], [19]
4	Transport	TCP/IP Stack	[20], [21], [22]
3	Network	IPv4, IPv6, ICMP, IEEE 802.11	[23], [24], [25], [26], [27], [28]
2	Data Link	CDP	[29]

TABLE 2.1: Protocols and corresponding layers of the OSI model, that are currently under research.

Across all fingerprinting techniques, the particular methods are classified as active or passive methods of generating and measuring fingerprint data.

2.1.1 Active Fingerprinting

Active Fingerprinting can be characterized as an active or intrusive way of generating fingerprinting data. The standard way to identify a target via the active fingerprinting approach is extracting data by collecting responses or reactions of a system or software from a set of predefined actions. The actions that are most valuable for fingerprint are those who are generating the highest entropy reactions. Like with biometrical fingerprints, a digital fingerprint is checked against a database of known signatures to calculate the degree of similarity and correlate known fingerprints or properties. Since ideal fingerprints are hardly achievable in a real-world scenario, because their corresponding stability and diversity properties are prone to false positives, the forensic value is generated by correlating multiple data sets across multiple properties. That ensures that the entropy of the generated fingerprint is high enough to identify a specific system, software, or user.

```
[root@machine ~]$ nmap 127.0.0.1 -O

Starting Nmap 7.80 ( https://nmap.org ) at 2020-06-03 11:59 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000072s latency).
Not shown: 999 closed ports
PORT    STATE SERVICE
631/tcp open  ipp
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6.32
OS details: Linux 2.6.32
Network Distance: 0 hops

OS detection performed.
Nmap done: 1 IP address (1 host up) scanned in 1.86 seconds
```

FIGURE 2.1: Active OS Fingerprinting with Nmap on localhost where a Linux kernel 2.6.32 is detected.

Figure 2.1 shows the scenario of active fingerprinting with the network testing tool Nmap [30]. Nmap is a widely used tool known for its scanning

and fingerprinting capabilities. One precisely working feature is Nmap's remote OS detection, which makes use of the TCP/IP stack active fingerprinting technique. For that, a series of UDP and TCP packages are sent to the remote host to perform multiple tests and subsequently collect and examine the responses. Every bit of the TCP packets received, e.g. the packets *initial window size, time to live (TTL), packet size*, are compared to the *nmap-os-db* database, where a lot of signatures have been collected. This method is efficient since each property gathered from the remote host is specific to a component of a standard TCP/IP implementation. Nmap stores more than 2600 different signatures of TCP/IP implementations that are classified by the vendor (e.g., Microsoft), OS (e.g., Windows), and device type (general purpose, router, switch, game console). An OS detection scan is fast compared to other script or service detections scans and can be highly useful for tasks where one must quick and remotely find out which exact OS and version is running on a system [30].

2.1.2 Passive Fingerprinting

In contrast to active fingerprinting, passive fingerprinting is a way to gather information about a service or system passively. In a client-server scenario, where a system or third-party receives messages from a target, fingerprinting can be done much stealthier.

```
[root@machine p0f-3.09b]$ ./p0f -i tun0

--- p0f 3.09b by Michal Zalewski <lcamtuf@coredump.cx> ---
[+] Closed 1 file descriptor.
[+] Loaded 322 signatures from 'p0f.fp'.
[+] Intercepting traffic on interface 'tun0'.
[+] Default packet filtering configured [+VLAN].
[+] Entered main event loop.

.-[ 130.149.39.0/44511 -> 88.221.214.0/80 (syn) ]-
|
| client   = 130.149.39.0/44511
| os       = Linux 2.2.x-3.x
| dist     = 0
| params   = generic
| raw_sig  = 4:64+0:0:1460:mss*44,7:mss,sok,ts,nop,ws:df,id+:0
|
`----
```

FIGURE 2.2: Passive OS fingerprinting by analyzing received traffic on the tunnel interface tun0 with p0f where a Linux kernel between version 2-3 is detected.

Figure 2.2 shows the tool p0f that utilizes purely passive traffic fingerprinting techniques, in this case, TCP/IP stack fingerprinting, to identify the systems behind any occurring TCP/IP communications [31]. When monitoring the tunnel interface *tun0* of the local system, it detects inbound traffic packets

sent from an IP with a *Linux kernel 2.2.x-3.x* and recognizes its unique signatures. The fingerprinting of the operating system also works for the passive approach, because the properties for fingerprinting are contained in the TCP packets received by the client. As we have seen in Figure 2.2, when a system sends packets to another system, the receiving system collects the data without actively probing the target on which fingerprinting should be performed. However, the probability of successfully identifying a sending target is usually much lower compared to the active approach since it lacks actively probing for high entropy answers. The signatures collected can be checked against the same database since the properties to check are identical.

2.2 Web-Device Fingerprinting

In contrast to the straightforward approach of categorizing the fingerprints of the operating system based on its active and passive fingerprint capabilities, taking fingerprints from web devices is a more complex field. First, taking fingerprints from web devices consists of several techniques to collect information about a target system and then correlate who is using that system. Like OS fingerprinting via the TCP/IP stack on the network layer (OSI 3), web device fingerprinting also targets information about the operating system, among other things. In contrast to Nmap, this is done on the application layer 7 of the OSI model, where browsers and other applications are running. For ease of use, web browsers must perform many privileged tasks, such as rendering objects, that require it to communicate with hardware components, such as graphics cards, or playing sound from music or video that requires it to interact with available devices such as speakers. However, this capability, as shown in Figure 2.3, can be misused to perform fingerprinting tasks using the web browser as an interface to interact and extract information from the underlying system [32].

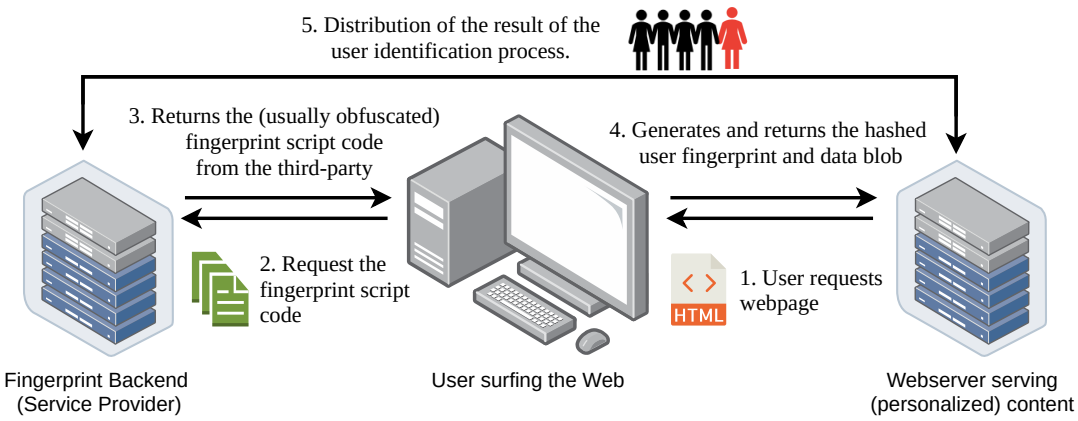


FIGURE 2.3: Fingerprint generation and exchange procedure to serve personalized content to the user.

First, when a user opens a browser, the browser DOM is initialized and the JavaScript engine starts. A user then requests a website that he or she wants

to visit. The webserver then provides the content by sending the HTML to the user. Once the user receives the HTML from the web server, all content such as images and third-party scripts are dynamically loaded into the page DOM. Within this time frame, all dynamically loaded scripts are executed, including fingerprint scripts. The scripts now generate a fingerprint without the user's consent. This collected private information is sent to an external server or fingerprint backend either with the next POST request from the user or as an additional request. The backend then correlates the fingerprint with a signature database and makes the result available to partner companies and websites in the network. These websites are now able to send targeted advertisements, job offers, or other relevant content to the user. That approach can be considered as both an active and a passive technique since the following tasks fulfill properties occurring in active as well as in passive fingerprint tasks.

- Controlling JavaScript and the DOM without consent of the user.
- Using system API's to use external devices.
- Injecting third-party code into the DOM of a target.
- Collecting user behavior e.g., time, clicks, logins.

2.2.1 DOM, Javascript and JIT

JavaScript is a just–in–time (JIT) compiled programming language that is widely used to enable interactive websites on the World Wide Web. JIT enables the browser to compile JavaScript code at runtime, thereby increasing the overall performance of a website [33]. Each browser starts its own instance of a JavaScript engine that features a JIT engine, able to execute the embedded JavaScript code on the client's computer.

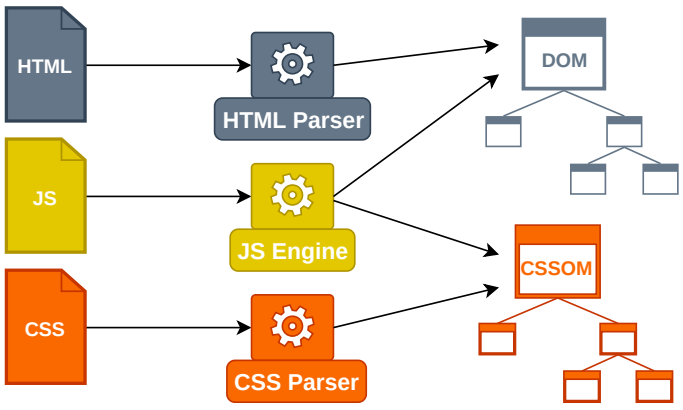


FIGURE 2.4: The main browser components DOM and CSSOM have a dependency on the JavaScript engine that enables interaction with various browsers API's.

When JavaScript runs in the context of a web page, it has the functionality to interact with its environments, such as the page DOM, CSSOM, or various

browser APIs [34]. For example, to read the current screen size of the browser window, we can execute the call *window.screen*, or *call navigator.plugins* to retrieve the list of installed plugins. While this can be very convenient for making a web page interactive and providing useful features, it can also be used to create a digital fingerprint for an individual device. Since JavaScript code is executed on a user's machine, the client theoretically has insight into the code running in his own DOM. As a result, a user may be able to copy ideas or parts of the actual JavaScript code itself from a company, even if it is the property of another company. To counteract this, many companies disguise their JavaScript code before it is loaded onto a user's computer. This obfuscation consists of restructuring the code so that it is intentionally made unreadable and it is very difficult to identify its exact functionality. The following Figure 2.5 shows an example of JavaScript code that has been obfuscated with the online obfuscation service *obfuscator.io* [35] versus the same code in plain JavaScript.

```
// Plain JavaScript
function hello() {
  console.log("Hello World!");
}
hello();

// Identical JavaScript functionality; obfuscated
var _0x3db2=['Hello\x20World!'];(function(_0x330efb,_0x3db29a){
var _0x5db9e5=function(_0x594477){while(--_0x594477){
_0x330efb['push'](_0x330efb['shift']());}};_0x5db9e5(++_0x3db29a);}
(_0x3db2,0x13a));var _0x5db9=function(_0x330efb,_0x3db29a)
{_0x330efb=_0x330efb-0x0;var _0x5db9e5=_0x3db2[_0x330efb];
return _0x5db9e5;};function hello(){console['log']
(_0x5db9('0x0'));}hello();
```

FIGURE 2.5: Hello world example of obfuscated versus not obfuscated code in JavaScript that showcases the inability of a user to read third-party code.

When visiting a website a user agrees implicitly to download the code of any script required and loaded by the website. The site is then able to execute any obfuscated JavaScript code without the explicit consent of the user. With the ability to access system APIs on the client, the fingerprint script in the browser provides a powerful way to perform active and passive fingerprinting on the client machine.

2.2.2 Cookies and Storage

An HTTP cookie is a data element that is stored by the storage mechanism of the client's browser to store client-specific data for a website. Many websites improve their user experience by storing various types of cookies on the client's computer [36]. Cookies are mainly used for the three following purposes [37].

- **Session Management**
  Logins, shopping carts, anything the server should remember.
- **Personalization**
  User preferences, themes, and other settings.
- **Tracking**
  Recording and analyzing user behavior.

Modern websites usually use authentication cookies to allow intuitive session management. Once a user logs in to a website using a web browser, that browser sets a session management cookie, so that the user does not need to log in again within the time the cookie is valid. In addition, websites use tracking cookies that track user behavior and surfing habits. These are usually linked to specific user accounts or the corresponding authentication cookies [36].

Cookies were once used as general client side-storage. While this was legitimate when they were the only way to store data on the client, it is recommended nowadays to prefer modern storage APIs, as seen in Table 2.2. Cookies were sent with every request, so that they worsen performance, especially for mobile data connections[37]. The current way to implement this is an API called *WebStorage API* and *IndexedDB*, which is accessible via a JavaScript call to *localStorage* and *sessionsStorage*. [36], [38]

API	Data Model	Persistence	Browser Support	Sync/Async
File system	Byte stream	device	52%	Async
Local Storage	key/value	device	93%	Sync
Session Storage	key/value	session	93%	Sync
Cookies	structured	device	100%	Sync
WebSQL	structured	device	77%	Async
Cache	key/value	device	60%	Async
IndexedDB	hybrid	device	83%	Async
Cloud Storage	byte stream	global	100%	Both

TABLE 2.2: Access to storage API's of a browser that are supported in current browser versions and used for saving fingerprint data.

Storage mechanisms play an essential role in the process of obtaining fingerprints. To exchange the generated fingerprint, the data is usually first saved temporarily using the browser's storage API. When a specific event is then triggered, the fingerprint is then loaded and sent to the associated endpoint. This is usually done either via a parameter in an e.g. Login-POST request or via an individual request to a separate third-party endpoint for fingerprint collecting.

2.2.3 Categories

Fingerprint scripts extract user behavior and user system information across a variety of fingerprint attributes and categories. A fingerprint category classifies a subject or component to collect information with high entropy about

the underlying system. According to *amiunique.org* [39], the following categories are the most used fingerprinting categories available via JavaScript.

- **UserAgent**
  A string giving details on the browser and its underlying operating system.
- **Canvas**
  Rendering of a specific picture with the HTML5 Canvas element following a fixed set of instructions. The picture presents some slight noticeable variations depending on the OS and the browser used.
- **WebGL**
  Rendering of a specific picture with the HTML5 Canvas element and the WebGL API following a fixed set of instructions. WebGL parameters : vendor, renderer, colors, textures, vectors, etc.
- **Flash**
  Test flash attributes for getting a list of different properties for browser and operating system.
- **Fonts**
  Test JavaScript attributes for getting an incomplete list of fonts installed on the operating system.
- **Language**
  Returns information about content, browser and system language settings.
- **Screen**
  Returns information e.g. width, height, depth, top of the screen.
- **Audio**
  Returns supported audio formats for the audio element. Provides real-time frequency and time-domain analysis information and audio elements rendered via an audio context.
- **Storage**
  Information on the support of local/session storage and IndexedDB.
- **Plugins**
  Calls browser-populated JavaScript attributes that gives the list of activated plugins in the browser.
- **Permissions**
  Permissions of the browser and sub components.
- **Media Devices**
  Returns the list of media devices present, e.g. microphones, cameras, headphones.

The category *canvas*, for example, gathers information about the system related to HTML canvas elements and their properties. In contrast, the category *audio* collects information about HTML *OscillatorNodes* and HTML *FrequencyAnalyzer* and calculates a collective value that can be interpreted as

audio fingerprint. Multiple fingerprinting attributes can relate to the same or multiple categories and are dependent on the fingerprinting implementation of the company. One of the publicly known and widely used open-source implementations of a fingerprint is the fingerprint service *fingerprintjs2* [40]. To analyze how a real set of fingerprint data behaves and looks, we let fingerprintjs2 generate a canvas fingerprint on our local machine.

FIGURE 2.6: Configured canvas fingerprint element created by the fingerprinter fingerprintjs2 (see Appendix B.1) that is stored and transmitted as BASE64 encoded PNG image. This image is intentionally somewhat blurred, as the variations of this rendered image increase the bits of entropy and thus the uniqueness of a canvas fingerprint.

Figure 2.6 shows the HTML canvas element configured with various settings and represents a real canvas fingerprint of fingerpintjs2. Table B.1 from Appendix shows all other 28 properties of the fingerprint information generated by fingerprintjs2 and their corresponding return values including the generated PNG from 2.6 encoded in base64 [41].

2.2.4 Detection Mechanism

Fingerprint scripts like fingerprintjs2 probing available fingerprint information by iterating over all accessible fingerprint parameters and storing the return values and any errors that occurred. If the check is successful, the script generates and extracts the fingerprint information.

```
var getCanvasFp = function (options) {
    var result = []
    var canvas = document.createElement('canvas')
    canvas.width = 2000
    canvas.height = 200
    canvas.style.display = 'inline'
    var ctx = canvas.getContext('2d')
    [...]
    if (canvas.toDataURL) {
        result.push('canvas fp:' + canvas.toDataURL())
    }
    return result
}
```

FIGURE 2.7: Creating a canvas fingerprint in fingerprintjs2 [41] by setting various parameters, which is then extracted via a data URL

In this case of a canvas fingerprint a simple canvas element is created. Then the code checks whether the two canvas contexts *canvas.getContext()* or *canvas.getContect('2d')*, which are required for the creation of a fingerprint, are also present, before they are used to create a high entropy canvas fingerprint. The fingerprint data collected is then saved to a collector variable where all fingerprinting data is joined, as shown in Figure 2.8.

```
var components = [
    { key: 'userAgent', getData: UserAgent },
    { key: 'webdriver', getData: webdriver },
    { key: 'canvas', getData: canvasKey },
    [...]
]
Fingerprint2.get(options, function (components) {
    [...]
    var data = x64hash128(map(newComponents, function (component) {
        return component.value }).join('~~~'), 31)
    }
}
```

FIGURE 2.8: All available fingerprint information of the target system is joined to a single collector variable which is then hashed to create a distinctive signature of a fingerprint.

In addition to data collection, fingerprint scripts calculate a hash value using a cryptographic hash function to provide additional value to the fingerprint. A hash function is a one-way function that is computationally infeasible to invert and can map data of arbitrary size to fixed-size values [42]. This means that the collected fingerprint data is uniquely associated with the calculated hash value. A fingerprint hash can, therefore, be treated like the signature of a system and has the property that only the system can recreate the same fingerprint hash [42]. The slightest difference between the calculated fingerprint data from one script would return a different fingerprint hash, which means that another system or system setting is present. This way, it is possible to quickly check on the server-side, whether two fingerprints are identical, without having to compare all individual attributes of the fingerprint data.

2.2.5 Verification

Since fingerprints are generated on user-controlled browsers, companies cannot assume that the generated fingerprint data are valid. Users could potentially interfere with fingerprint scripts and manipulate the technology used to generate fingerprints. For this reason, many fingerprint scripts perform cross-comparisons between different fingerprint attributes to extract the same data from different features and detect tampering. Another way to verify the data is to additionally test for attributes that are data specific. Fingerprintjs2, for example, performs verification checks on easily forgeable values, such as *language, resolution, operating system, browser*. The following function

from fingerprintjs2, which is part of the *construct* metric in Figure 2.9, is a prime example of this scenario.

```
var getHasLiedBrowser = function () {
    // Get attributes from value and do cross checks for verification
    var userAgent = navigator.userAgent.toLowerCase()
    var productSub = navigator.productSub
    [...]
    // Eval the string length and return true if tapering is detected
    var tempRes = eval.toString().length
    if (tempRes === 37 && browser !== 'Safari' && \
    browser !== 'Firefox' && browser !== 'Other') {
      return true
    }
    [...]
}
```

FIGURE 2.9: Checking for manipulation attempts in the User-Agent property in fingerprintjs2 [41], where simple and unique browser-internal differences can be used to determine which browser is present and whether any manipulation has occurred.

Another standard method of testing attributes such as the string *userAgent* is to test this data against other browser-specific properties for validation since the property consists of several sub-properties. If a fingerprint script detects a manipulation attempt, the collection function returns *true*, and the script continues to treat this attribute as a lie. Another possible second way fingerprint2.js can detect if a browser or *userAgent* string has been tampered with is to actively generate errors. By e.g. throwing an undefined exception *a* and checking the error with the method *err.toSource()*, fingerprint2.js is able to distinguish between different browsers and thus recognize that someone has manipulated the fingerprint data.

Even if fingerprint scripts that verify the collected data are validated, there is great uncertainty as to whether such data are accurate. The default method to work around this problem is to bind the fingerprint to a session. That way, a user logs in and the validity of the fingerprint can be calculated based on previous fingerprint calculations and the newly set cookie, as we see in Section 5.1 of the evaluation. In fact, companies still have considerable problems distinguishing between a user who has updated his system (software or hardware) and a user who is completely new. To this point of research, no reliable and comprehensive method is known that would allow an organization to safely re-identify a user after changing its hardware or software components. Fingerprints are currently most effectively used to protect websites from bots [43].

2.3 Field of Application

In this section, we show the different uses and applications of fingerprint techniques and show their state of current research.

2.3.1 2FA and Fraud Detection

Fingerprinting can be effectively used to detect and indicate different types of fraud. Malicious users can gain access to a victim's sensitive information from sources, such as data breaches, phishing, or malware, and use this information to take unauthorized actions. Fraud attacks are a form of identity theft where, in most cases, the user does not even know that sensitive data has been compromised or stolen [44]. Being able to distinguish between normal and unusual behavior of client machines, which may be bots or fraudsters, can be a very valuable piece of information for financial institutions or anyone else where customer authorization is critical. Many web services already implement two-factor authentication, which sends a notification and a key to the users via a second communication channel such as e-mail or SMS. The device is then registered by the company and assigned to the user internally.

There is a lot of discussion about whether this has a high negative impact on the usability for the user and whether this method is a safe way to have a mechanism to validate an existing user [45]. For that reason, companies are implementing and experimenting with a soft second factor and web device fingerprinting [46, 47, 48, 49]. The user has to reauthorize if inconsistencies in his software or hardware configuration are detected by web device fingerprinting. The new fingerprint is then associated with the user session or device, and the user can log back in using that device.

2.3.2 Ad-Networks

In most cases, a company uses third-parties for taking fingerprints. Third-parties are other companies whose business model consists of collecting information about users and selling this, in most cases behavioral information, to other companies. These advertising companies operate in large customer networks where they manage to create and maintain information databases about user behavior. For them, fingerprint information about the device used is very valuable because it can correlate with which user is associated with which activity [50]. For this reason, this information is even more valuable for potential third-party customers, for example, to display targeted information according to the detected behavior [51].

To use a fingerprint service, a website imports a fingerprint script via the HTML *href* tag and sets a trigger event, such as *page ready* or *login*. The script is then executed as soon as the trigger event is reached. The generated fingerprint is sent either directly to the third party or inline and covertly in the regular dialogue of requests with the website. However, the information about who passes this fingerprint data on to whom remains unknown

to the end-user. Advertising networks are used to collect and share information about users and their behavior on a single site or across multiple sites. Because the use of and reliance on cookies is threatened by certain privacy modes in a browser and the general ability to disable cookies, advertisers are increasingly turning to cookieless methods such as device fingerprinting [52, 14].

2.4 Countermeasures

To defend against arbitrary data collection by companies and their contractors, users have several options to choose from. Since the measurement of user behavior can be performed on many different layers of the OSI model, there is no general defense mechanism that disables all data collection and fingerprinting. In the following sections, we will show general methods that render at least some of the data collection mechanisms unusable.

2.4.1 Browser Based Protection

Within the web browser, the most effective measure against fingerprinting is to disable JavaScript itself. If JavaScript code cannot be executed, any fingerprint script will become non-executable, and no fingerprint can be generated. However, the big caveat is, that most websites rely on JavaScript being enabled. Standard functions, such as login mechanisms or dynamic loading of content on the website, are usually dependent on JavaScript and are useless if JavaScript is not available. Yet creating websites is still possible without the dependency on JavaScript [53, 54]. At least some companies, e.g. *Amazon* and *Twitter*, have decided to make their website available, although with reduced functionality [55]. A similar approach is to revoke the browser's ability to store cookies of all types through a cookie banner that allows a user to choose whether to accept the privacy policy of this website or to selectively disable all cookies in the browser settings. The idea is to disable the ability to attach a fingerprint to a session. Therefore, the generated fingerprint data will not be bound to an existing user account [52]. However, as with the deactivation of JavaScript, some websites rely on cookies for authentication. Because sessions are usually tied to authentication cookies, a user cannot log in and is redirected to an error page if all cookies are disabled. Perhaps the most sophisticated way to deal with third-parties and still have functioning websites is to use ad blockers. These are browser extensions that are available for each user to download from the browser app store. They operate by filtering, blocking, and disabling potential tracker and ad scripts. They are also able to disable outgoing communications to third-party services. However, ad blockers are limited to the ability to match against predefined third-party signature lists stored in signature databases [56]. Further, there are some browser settings that promise enhanced privacy if enabled.

- private browsing
- disabling specific cookies (e.g. social media, third-party)
- disabling tracking content
- disabling cryptominers
- disabling fingerprinters
- enabling DoNotTrack flag

The browser sets a flag, e.g. the *DoNotTrack* flag, that is accessible for a website via the browser API and suggests when enabled, that the current user does not want to be tracked. Those settings are usually not disabled by default and offer only a small description for their use case. Privacy-conscious users can choose to enable them in the Firefox web browser, but these settings do not address the underlying privacy concerns and instead cover only a narrow scope [57].

2.4.2 Network Based Protection

For disabling network interactions with third-party ad-networks, solutions such as pi-hole are a popular way to protect even entire networks from tracking [58]. They include custom DNS and DHCP servers to filter network traffic and restrict connections to known ad-networks with black and white lists. Disabling fingerprint activity across the network layer is, however, only partially successful. Fingerprints can also be sent directly through non-third-party websites, which should not be blocked in this case and are, therefore, not covered in this scenario.

2.5 Related Work

To this day, numerous methods have been created to generate and extract fingerprints from devices and to actively collect, monitor, and identify users. Due to many companies building up large infrastructures to collect fingerprinting information at scale, projects like the Panopticlick Project [59] by Eckersley *et al.* [60] and the Electronic Frontier Foundation [61] showed the impact of user tracking to the world by broadly analyzing fingerprinting for the first time. Eckersley's and EFFs initial research on fingerprinting technology was followed by extensive research in this field, as described by Ekersley [60], Mayer [62], Acar *et al.* [63] and Vastel *et al.* [64], where issues, such as the anonymity on the Internet, the uniqueness of a web browser together with persistent tracking mechanisms, and the privacy impact were questioned and brought to the public for discussion. Technologies such as *evercookie* emerged and initiated research such as that conducted by Acar *et al.* [50] including the discovery of respawning or cookie synchronization, which allowed websites to reload a user's stored cookies even if the user had deleted their local cookie store. Through the instrumentation of a browser

and the analysis of canvas fingerprints on the home pages of 100,000 websites, they have been able to identify 5.542 websites that use fingerprinting. From then on, the increasing demand for defense mechanisms against fingerprinting resulted in further extensive analyses such as by Mozilla [65], Wu *et al.* [66], Trickel *et al.* [67], and Laperdrix *et al.* [68]. The researchers challenged client-side diversification by randomizing the core browser objects to render device fingerprints unusable and introduced new features in web browsers to counter current techniques, such as the then-emerging WebGL-based browser fingerprinting. At the same time new ideas for the use of fingerprints were developed, e.g. by Takei *et al.*, who proposed a method to only use cascading style sheets for fingerprinting web devices [69], and Mowery and Shacham introduced advanced canvas fingerprinting in HTML5 [70]. Mowery and Shacham proposed a method that allows any browser running JavaScript to generate consistent and highly entropy fingerprint information from a simple rendered phrase using differences in system fonts within an HTML Canvas element, and therefore differentiate between different WebGL renderers and hardware, such as graphics cards. Furthermore, already existing approaches, as described by Vastel *et al.* [14] and Starov [71], were further enhanced, and the trend of fingerprinting discussed. According to Snyder *et al.* [72], the expansion of browser and device functionality also increase the unintentionally and newly introduced ways of identifying users. In order to improve the privacy and security of users, they first introduced a browser extension that allows the user to selectively restrict access to low-benefit but high-risk browser features, thus creating a user-visible choice of benefits in terms of personal privacy protection. For the same reason of giving users a choice of their privacy settings, current online tools, like AmIUnique [39], FingerprintJS2 [40], and Panopticlick [59] were brought to life. These tools also raise user awareness and show in detail how far we have come by visualizing the user's current fingerprint and enumerate configurations entropy and risks for each fingerprint value detected. Recent work by, e.g. Vastel *et al.* [73] and Azad *et al.* [74] show that fingerprinting finally emerged into several niches of applicability beyond collecting fingerprint for target advertisements. Their studies show that websites are now able to detect the automated use of their service by using fingerprinting techniques for bot detection. Unlike CAPTCHAs, which are the popular method for detecting malicious crawlers and entities, fingerprint recognition requires no user interaction and is not influenced by advances in automatic image and audio recognition research [75]. On the other hand, they investigate the development and use of anti-fingerprinting browsers which is actively being used to evade detection by malicious actors mimicking browser environments of victims for harmful purposes [74]. This year, Laperdrix *et al.* [76] examined newly developed fingerprinting techniques and analyzed specific attributes in the browser ecosystem that can be exploited for fingerprinting, and compared them to existing defenses, including the Tor browser [43], EFF's Privacy Badger extension [77], and Mozilla's Enhanced Tracking Protection [65]. The investigation has shown that, in terms of protection against a wide range of threats, protection mechanisms are usually limited in scope and can be

easily circumvented by making minor changes to the threat or its code. To protect the privacy of users, data protection regulations such as the General Data Protection Regulation (GDPR) of the European Union attempt to regulate the use of private information in digital services by restricting the access and distribution of this data [78, 6]. The EU has also begun the process of updating its ePrivacy Directive, best known for its mandate that websites must warn about any cookies they are using. Although it is still a major challenge for all participants to delimit the legitimate interest and collection of fingerprints by a law that restricts companies from misusing personal user data [6], this work combines several approaches from research from the past to address the current issues.

As described in Section 1.3, this thesis examines fingerprinting techniques and investigates how widespread web device fingerprinting has become. Like Snyder *et al.* conducted in [72], we develop a browser extension as a tool that further enhances the visibility of fingerprinting and transparency to the user. In contrast to Snyder's approach of allowing the user to restrict certain privileges on the application, we follow the methods of AmIUnique [39], FingerprintJS2 [40], and Panopticlick [59], providing the user with full accessibility and transparency regarding the fingerprint activities currently being executed. Furthermore, in contrast to Takei *et al.* [69], Wu *et al.* [66], Acar *et al.* [50], Azad *et al.* [74], Mowery and Shacham [70], who in their studies examine individual parts of a fingerprint or the browser to perform fingerprinting on, we extend the scope of application for fingerprinting to the entire spectrum of fingerprints known to us. Along with open source fingerprinting services, such as FingerprintJS2, we are creating a monitoring tool that is capable of covering a variety of sub-fingerprinting activities, including canvas, WebGL, and audio. Our tool displays the current fingerprint activity in a pop-up window, increasing privacy awareness and transparency for its users. Beyond Vastel *et al.* [73] investigations that inspect the web and probes websites that detect automated services like crawlers and bots, it is possible to learn more about these services. To achieve this, our monitoring software tracks the behavior of each fingerprint script on each site in order to further evaluate its aggressiveness, scope, and possible affiliations with other websites. Since we have access to real-time fingerprint activity, we perform our analyses on various defensive browser extensions, like Laperdrix *et al.* examined several defensive browser extensions in [76]. However, our approach examines the effectiveness of these defensive extensions in detail and also provides immediate feedback to the user as on which extension restricts which fingerprint actions.

Chapter 3

Design

This chapter introduces the concept behind the fingerprinting monitor FPMON, which is the main contribution of this thesis. It gives the reader an overview of the approach, functionality, and architecture of FPMON and explains the decisions that were made in the conceptual phase of the development of this software.

3.1 Fingerprinting Monitor

FPMON is a tool for the passive monitoring of fingerprints in real-time while surfing the Internet. The idea behind FPMON is to allow users to gain insight into the code activity of websites and potential privacy concerns without any specific domain knowledge of programming languages and browser features. The extension is designed for chrome-based web browsers, such as Chromium or Google Chrome browsers, as these had the highest market share of > 65% of all web browsers in use in 2019 [79]. In addition, FPMON is easily portable, with little or no change to the browser-specific extension syntax, but not to the logic of FPMON itself, to be usable with other browsers such as Firefox and Safari. The main goal of this tool is to enable users to evaluate websites themselves. For this purpose, FPMON provides an overview that indicates unusually high fingerprint activity, e.g. on payment pages, which could be an indicator of potentially unsafe behavior and risk when using credit card information on that site. This is accomplished by monitoring JavaScript in the browser itself.

Having control over the programming language used and the environment in which it runs provides all the capabilities needed to monitor fingerprint behavior. FPMON collects information about a defined set of attributes and functions related to fingerprint activity. For each of these attributes and functions, the functionality of a single attribute or function is overwritten. In this way, we append a callback mechanism to the original functionality, which triggers the execution of additional logging code. Using this method, the following information can be obtained from the calling function or attribute by FPMON

Name of function or attribute	fillText()
Argument(s)	{"0":"Cwm fjordbank glyphs vext quiz,"1":4,"2":45}
Return value(s)	None
Number of calls	3
Related fingerprint category	Canvas
Location in the order of all calls	[...]Canvas;Canvas;JS_fonts;**Canvas;**[...]

TABLE 3.1: Information collected from a single canvas fingerprint call that FPMON has real-time access to.

With the name of the recognized function or attribute, the call can be identified by FPMON and assigned to a fingerprint category. Collecting the function and attribute arguments gives a more detailed insight into the exact actions the fingerprint performs. Return values contain a lot of information about the actual fingerprint behavior, as they contain the actual fingerprint data. However, it should be kept in mind that return values can be complex structures containing the fingerprint data, but it is difficult to find out the exact data of the fingerprint information. The recording of the number of occurrences together with the sequence of functions and attributes called not only provides information on how often fingerprint calls are made but also shows the scope of a fingerprint's activity since repetitive patterns can be measured.
In addition, FPMON distinguishes into two different groups of characteristics, which differ in their intrusive behavior.

- **Sensitive Features** are often used in an intended manner. They are necessary to improve the user experience, for example, to display the correct language or to store the session state. To identify a user with high accuracy, several sensitive features must be used simultaneously.
- **Aggressive Features** are features that are often used in an unintended way. They can typically create a high level of entropy and make use of browser features that are only very rarely used in practice on the site, such as your battery status or WebGL. Many of them can identify a user without requiring other functions.

After collecting function names and attributes, FPMON categorizes them into the following 40 different aggressive(18) and sensitive(22) fingerprint categories.

Aggressive Categories:

- Canvas
- JavaScript Fonts
- List of Plugins
- WebGL
- Webdriver
- Audio and Video Formats
- Audio
- Media Devices
- Frequency Analyzer
- Geolocation
- Connection
- App Version
- Permissions
- Product Sub
- Oscpu
- Battery Status
- Device Memory
- Hardware Concurrency

Sensitive Categories:

- Product
- Vendor
- Vendor Sub
- App Code Name
- App Name
- Storage
- Drag and Drop
- DoNotTrack
- Platform
- CPU Class
- Mobile
- Screen and Window
- Cookies
- Java
- Flash
- Timezone
- Content Language
- Browser Language
- System Language
- Online
- UserAgent
- Build ID

If users want to add the extension to their browser, they simply need to load the extension manually using the "Load External Extension" function of their browser. Once the extension is loaded and active, the monitor passively detects fingerprint activity without a noticeable impact on web browsing performance. We tested this on may websites hosted on localhost and web and found out that the performance differences were depending on the size of the website between 10 to 100ms. This difference is smaller than the normal load time variations a website and its third-parties have while browsing. Without interacting with the extension by clicking on it, as shown in the following Figure 3.1, only a small icon to the right of the address bar is visible, displaying the current fingerprint activity of the site in the active tab.

FIGURE 3.1: FPMON icon color change at specific thresholds to indicate the amount of present fingerprinting activities.

The symbol and text of the badge will change depending on the aggressiveness that the FPMON extension has observed for a website. A very active as well as aggressive fingerprint activity is indicated by a red fingerprint icon, while yellow, green, and gray indicate medium, low, or no activity. In addition, the rating of the detected fingerprint categories is displayed as a badge text of the icon. This allows users to quickly determine if fingerprinting is currently active in their browser. When a user clicks on the FPMON extension icon, a small popup window will appear (see Figure 4.4) that displays more details about the currently recognized fingerprint calls. The information displayed includes the monitored domains, how many features are detected in how many categories, and the list of active sensitive and aggressive categories.

3.2 Fingerprinting Crawler

For automated analysis, we present another tool we develop, the fingerprinting crawler (FPCRAWL), which can be used for large-scale studies with the tool FPMON. This tool builds on the core of the FPMON browser extension and makes it possible to apply the monitoring techniques of FPMON to a huge list of websites. Like a wrapper for FPMON, FPCRAWL processes input and output automatically. For this, the web browser is instrumented in an automated fashion with the use of the Selenium framework [80], which provides a way to test software on pre-configured different browsers and versions and visits a list of pages and records their behavior.

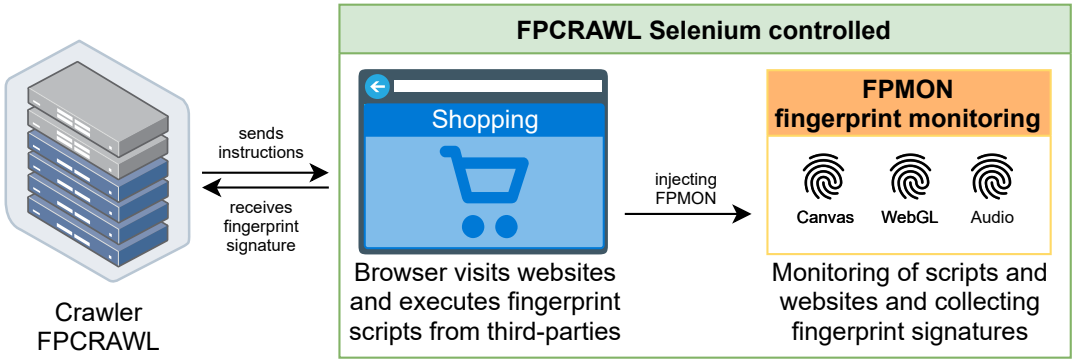


FIGURE 3.2: Generation and extraction of a fingerprint signature within a docker node that is controlled and orchestrated by FPCRAWL via Selenium.

With Selenium, the crawler iterates over a list of web pages and navigates to each of the sites using a modified version of FPMON. The extension had to be modified to correctly handle synchronization and timeout issues and send the collected data to a local server using HTTPS requests. This local server logs all received data in a simple database that can be used later for further analysis. The crawler is able to scan websites with a certain depth by extracting all links with the same domain based on the URL specified via the input list. To speed up the time needed for the scanning process, we run several instances of the browser node, since the measurements are independent of each other. In addition, orchestration for large-scale scans is done via Docker, which provides a way to run applications safely isolated in a container and namespace, packed with all its dependencies and libraries. With Docker and its Docker-compose tool, which enables the execution of applications with multiple containers in user-defined networks [81], the crawling process can be performed quickly and error-prone and is easy to set up in a virtualized environment.

The biggest difficulty for the crawler was to choose a reasonable configuration threshold for timeouts and scan depth, as we cannot predict when and where the fingerprint is scheduled or needs to be triggered by an event. In general, we have greatly improved scanning time by adding basic signaling between the browser and crawler for various corner cases, such as DNS or HTTP 404 errors, and thereby reducing the overall scanning time.

3.3 Architectural Concept and Evaluation

The typical way fingerprint architectures distribute the generated fingerprint data is usually very similar on all websites that implement fingerprinting. They wrap the data on the client machine into a request and send the data to themselves or third-parties. For this reason, our concept implemented in the FPMON extension is very effective and generally applicable in any scenario where JavaScript is executed and intercepted on a user machine. As in Figure 2.3, where the general fingerprint approach is shown, the following steps are performed by the FPMON to monitor a fingerprint of the user.

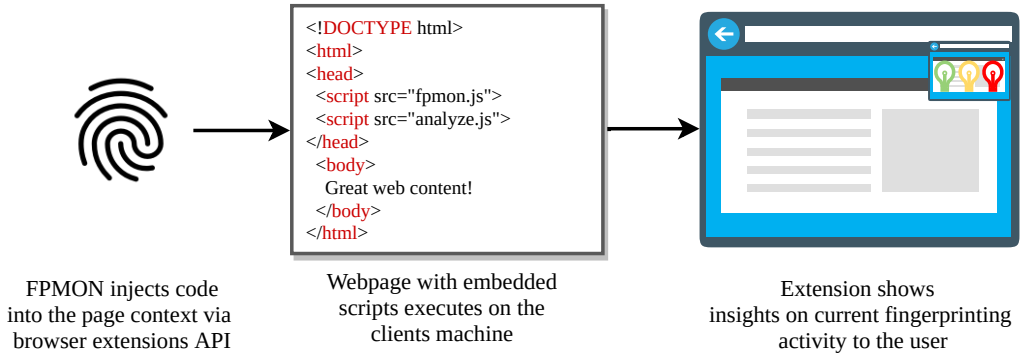


FIGURE 3.3: FPMON code is injected into the user's DOM, which is executed in addition to third-party fingerprinting scripts in order to monitor the behavior displayed in the extension popup.

The step where an HTML form dynamically loads external scripts and content is the entry point for our FPMON extension to monitor JavaScript execution. As shown in Figure 3.3, the browser performs a content injection to use our FPMON extension. This allows the browser to add a script node in the currently active DOM, just as a website does when it dynamically loads external resources. The difference here is, that our code injection is executed earlier than any externally loaded script. Therefore, it can override JavaScript functions and attribute calls before the website scripts start executing. This component is the reason why FPMON is so powerful and will not be noticed by anyone except the user using the extension.

Fingerprints are usually dependent on the script code being executed normally and not being touched by the user. However, fingerprints are dependent on everything being executed as intended to produce a valid fingerprint truthfully so, that it can be used for its intended purpose. Our method for monitoring fingerprint calls uses exactly this mechanism. We imitate the real behavior of the fingerprint by doing the same. We intercept the action and execute it immediately, as the normal fingerprint would have done instead. This gives us full control over every output and input the fingerprint script receives before it even calls the desired function. For this reason, the FPMON works for all JavaScript-related fingerprint scanners whose functions we track, and is an effective way to monitor browser and script execution

without the knowledge of a website. However, the FPMON extension is limited in terms of recording the activities of a fingerprint. Any fingerprint activity that is not covered by the FPMON extension, e.g. all home-brewed or custom actions, will not be considered by the monitoring process. In general, FPMON covers the standard and well-known actions of a typical fingerprint that we have found in open-source implementations, e.g. [40] to evaluate if and how aggressively a fingerprint works. Finally, FPMON monitors most of the fingerprint activity and forwards this information to the user to increase awareness which is the main goal of this software.

Figure 3.4 shows our crawling architecture with the FPCRAWL crawler. The concept of our crawling environment shows the key components in this architecture. First, the crawler starts a task with a user-supplied input list of sites or domains. Our crawler has full process control and can perform error-handling tasks, such as re-queuing unsuccessful list entries or restarting the entire Docker network including the Selenium hub and browser nodes.

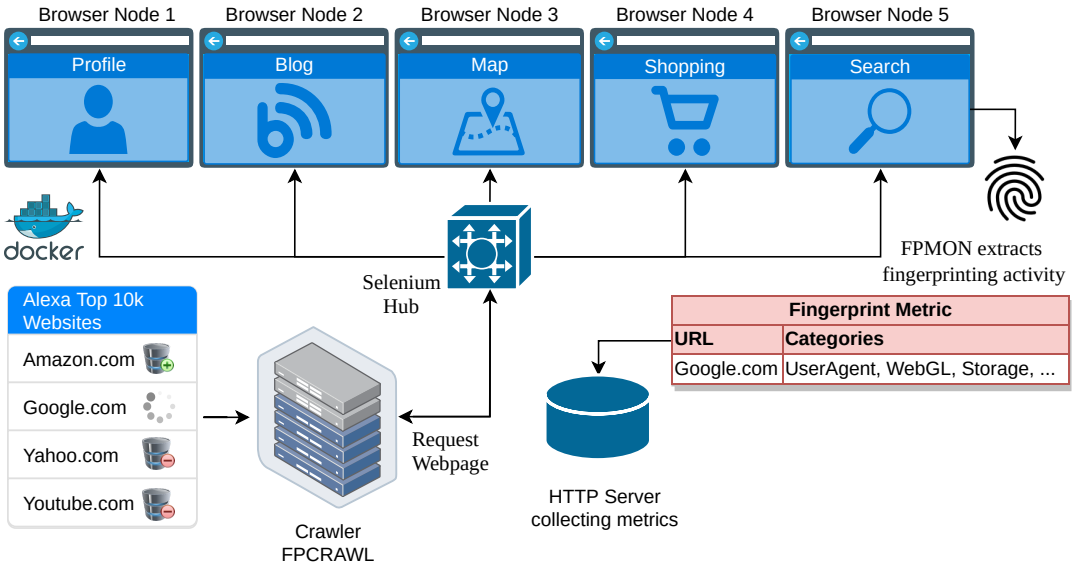


FIGURE 3.4: Architecture of FPCRAWL where each entity is a container or an image in the Docker environment to collect signatures for large lists of websites.

An important hub is the Selenium Hub. The hub manages every interaction between the browser nodes and the crawler itself. As long as this node can handle the spawning, login and logout events of the browser nodes correctly, to our knowledge and experience, there is no bottleneck other than the CPU and RAM usage during the scale-up. Our setup scales horizontally with Docker by adding more browser nodes. We used 14 concurrent jobs, evenly distributed across 14 cores on a machine with 40 gigabytes of RAM, to perform all our scans in Chapter 5. The key component of our Docker architecture is our FPMON Browser extension, which monitors and extracts the fingerprint behavior of each Docker Browser node. While conducting performance tests, we found that the monitoring logic of our FPMON extension works most efficiently when each browser node runs on an individual

core to minimize the time it takes a website to load consistently on average. However, the limit for extracting a fingerprint with FPCRAWL depends on the actual implementation of a fingerprint. The crawler visits landing pages and sub-directories as described in our first study in Section 5.1.1. However, fingerprinting activities that are tied to the functionality of websites, such as login or other authentication or session management, are not covered by the crawler because no fingerprinting activity can be monitored if no start event is triggered. Fortunately, we have largely measured fingerprinting activity whose fingerprint generation is based on the page load event and the only extraction is based on triggers such as login events. In addition, any fingerprint activity behind a clicked cookie banner is not captured by FPCRAWL. For users who perform manual fingerprint analysis using the FPMON, this does not affect the functionality or reduce coverage.

Chapter 4

Implementation

In this chapter, we will explain how we implemented the FPMON monitor and the FPCRAWL crawler and its docker environment. The goal is to give the reader a deep understanding of the methods and techniques used to build a tool for monitoring fingerprint behavior. We will show the technologies used in the development and identify the most difficult problems to solve when creating a fingerprinting monitor and a fingerprinting crawler.

4.1 Development Environment

The software was built and tested on Linux Manjaro (4.19.107-1) and a KVM LVM cloud instance of Linux Ubuntu 19.04 (5.0.0-13-generic). For the development environment of the crawler FPCRAWL, we use the Sublime Text 3 IDE with a Python 3.8 interpreter and additional packages required for Selenium (3.141.0). For testing the JavaScript code of the fingerprinting monitor FPMON, we use the browsers Chromium (80.0.3987.122 Arch Linux), Google Chrome (80.0.3987.132), and Mozilla Firefox (73.0.1). For orchestration, we use Docker (19.03.6-ce), docker-compose (1.21.0), and a Docker Chrome image (3.141.59-zirconium) available at Docker Hub or in the Selenium Git repository [81, 80].

4.2 Browser Extension FPMON

We have integrated the functionality of the fingerprinting monitor into a browser extension that uses the JavaScript content injection approach, where a script *content.js* is injected into the DOM of the current web page of a browser. We decided to use Chromium-based web browsers as browsers, which we want to instrument and perform our analysis. Chromium is a free and open-source software project from Google and forms the core of many different browsers like Google Chrome, Microsoft Edge, and Opera. All of these browsers use the same V8 JavaScript engine [82], which is the target of this research to extract fingerprint information. The base image of Chromium is more lightweight than e.g. Google's custom build Google

Chrome. Chromium itself only lacks some exclusive features, such as automatic updating, crash and error reporting, and some video and audio libraries and their licenses. According to Netmarketshare [79], Google Chrome has a market share of 68% in 2019, making it by far the most widely used browser for the web. For this reason, we decided to create the FPMON extension for Chromium-based browsers. FPMON is written in standard JavaScript, so no additional packages need to be loaded. The extension can also be loaded in browsers that are not based on Chromium, with only minimal changes to the extension structure. For the structure of the FPMON browser extension, we have used the following common browser extension architecture.

- **manifest.json**
  This file contains general information about the browser extension, such as the most important files and the capabilities the extension uses. In our case, this file tells the browser that we intend to use a content injection that has the scope to include all URLs visited, has permission on the active tab, and is able to connect to a server running and listening on the loopback interface on localhost. Most importantly, the file defines the time when the script and extension should be executed. FPMON is injected at document startup and executed before anything else is loaded into the DOM to be the first code and script to be executed.
- **content.js**
  This file contains the logic of the fingerprinting monitor FPMON and is explained in the next Section 4.2.1.
- **background.js**
  Background pages like our background.js are defined in the manifest file and contain event handling and message passing tasks. For FPMON, this file interacts with the popup.js extension script and sets various symbols and badge texts according to the score calculated by the content.js file. This greatly improves the user experience of the browser extension as users can see in real-time the increasing score of the badge text and the color changes of the icon.
- **popup.html**
  This file defines the structure of the extension page and is linked to its backend file popup.js. FPMON displays metrics, such as the URL, the coverage of the detected attributes and categories, and a list of sensitive and aggressive calls.
- **popup.js**
  This file contains the logic of the extension itself and communicates with the extension DOM, the content script, and the background page. In small time intervals, popup.js requests updates of the metrics calculated in content.js and updates the displayed information about the extension DOM.

4.2.1 Content Injection and Monitoring

Content injections are code injections into the active DOM of a web browser to improve functionality and usability. As defined in the manifest file, the logic of the FPMON is executed at the beginning of the DOM. However, the *document_start* attribute defines the entry point for all loaded scripts to be made executable. Therefore, all scripts that are loaded until this event is triggered are made executable simultaneously. This property interferes with our FPMON logic, since some third-party scripts would run faster and return faster than our monitoring code if we relied on it exclusively.

```
console.log("[content.js] injecting monitor.js into page context")
var s = document.createElement("script");
function inject() {
    // FPMON monitoring logic
    [...]
}
// Append an inject() function that executes faster than a
// dynamically loaded script via the href tag.
var actualCode = "(" + inject + ")()";
s.text = "" + actualCode;
(document.head || document.documentElement).appendChild(s);
console.log("[content.js] injections done")
```

FIGURE 4.1: Injecting the monitoring logic into the browser's active DOM and trick the DOM to execute our code first.

To overcome the difficulty of not detecting every call with the FPMON, we realize the injection via an additional *inject()* function. As shown in the code in Figure 4.1, we make sure that the content script is executed earlier than any other script (on the page or third-party JavaScript), waiting for the *document_start* event. To do this, the content script creates a new JavaScript element *script* in the DOM. It then appends the logic of the fingerprinting monitor as a self-calling JavaScript function to this node making it self-executable that calls itself at the very beginning of a DOM startup.

JavaScript is a single-threaded language, i.e. only one task can be executed at a time. Since our monitoring code itself is wrapped in a function, it, therefore, blocks the single JavaScript thread and ensures that all executed calls are covered and the desired fingerprint functions and attributes are bound as callbacks. Even if other scripts are ready for execution and run faster than our monitoring code, our injection function blocks the one available JavaScript thread until it finished overriding all defined functions for monitoring the fingerprint behavior. At any execution speed caused by a potentially slower user machine, our monitoring code has the privilege to interrupt the execution before the execution of another script starts. We have found that this technique eliminates most, but not all, of the recognition errors caused by *async* and *defer* tagged scripts. Those enable concurrency in a JavaScript thread which helps to increase the speed of the web application. The injection function now contains the code for the FPMON monitoring

logic. It performs two internal functions to override our defined JavaScript function calls and attributes that we want to monitor.

```
function overrideFunction(func) {
  // Overriding a function and appending monitoring logic
  // to the original function call
  func.obj[func.propName] = (function(orig) {
      return function() {
        // Monitoring logic
        [...]
        return orig.apply(this, args);
      };
    }(func.obj[func.propName]));
}
// Overriding the object prototype/method:
// window.AudioContext.createOscillator()
overrideFunction({propName: "createOscillator",
    obj: window.AudioContext.prototype})
```

FIGURE 4.2: Overwrite a function prototype and append the FPMON monitoring logic to create a callback function for each part of a fingerprint.

The function *overrideFunction()*, as shown in Figure 4.2, takes the name and object prototype of the function we want to monitor as an argument. The object *window.AudioContext.prototype*, together with the function *createOscillator()* is overwritten with a generic function prototype *function*, that acts as our callback function. This callback is executed instead of the original function, while technically it is the original wrapped function. The first thing the callback does is to execute the original function in order to get valid return values of the function call. This part is crucial to be executed in time and not to affect the behavior of the website.

```
  // Appending monitoring logic to an attribute call
function overrideAttribute(prop, subProp) {
  // Monitoring code
  [...]
  return window[prop][subProp];
}
// Overriding a object property
window[prop].__defineGetter__(subProp, () => {
  return overrideAttribute(prop, subProp);
});
```

FIGURE 4.3: Override an object property and append the FPMON monitoring logic to create a callback function for each part of a fingerprint.

For overriding browser attributes, we use the *obj.__defineGetter__(prop, func)*[1] method. This method binds the property of an object to a function to be

[1] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/__defineGetter__

called when that property is looked up. In our case this function is our callback function *overrideAttribute()*, as shown in Figure 4.3. As with the override functions, the next step is to call the original attributes again in the callback to ensure a valid return value to preserve the site functionality.

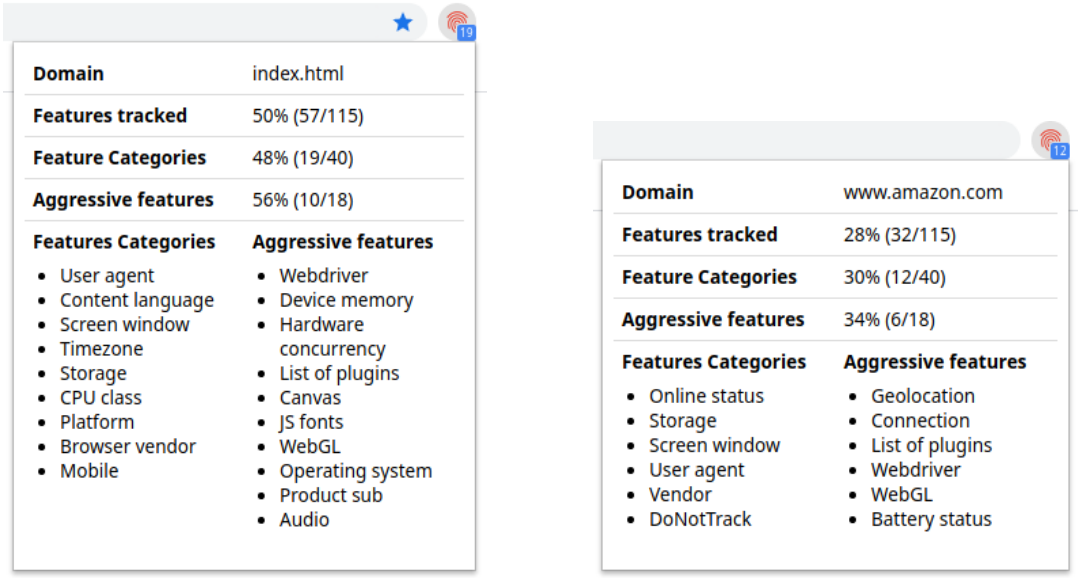
Within both callback functions, we implement an evaluation mechanism to monitor and collect all information about the original call. We create a global metric collector that tracks the recognized calls by name, arguments, return values, size of the return value, and call counter. This way, we continuously collect callback information, update the collector information as callbacks occur, and provide real-time monitoring data accordingly. We have chosen to have a collector separate from the actual extension logic in order to be able to use the data collected at a particular time or event. We evaluate the data and send the processed metrics to the extension's DOM, where the extension user can see the current fingerprinting activity.

The main issue we face here is to check that no other JavaScript code in the DOM of the browser exists that executes before our injected monitoring code. Together with the *document_start* event defined in the extension's manifest file and our custom injection function, we manage to achieve this goal. To validate this, we implement a method shown in Appendix A.1 to monitor all occurring events in the DOM, especially those related to fingerprinting in JavaScript scripts. Before anything happens in the DOM, even before our monitoring code is injected, we use an observer mechanism *MutationObserver()*[2] to control HTML script elements and check for any JavaScript activity. With this DOM observer technology, we are able to control and delay each script execution and determine that our monitor behaves exactly as intended. We also check scripts for their asynchronous capabilities and test for nodes that contain *async* and *defer* tagged script elements. To fix all errors that can be caused by asynchrony, we disable all *async* and *defer* script tags and treat them as if they do not have this property.

4.2.2 User Interface

Users who install our FPMON browser extension have access to monitoring information about the currently active websites they visit. According to our performed tests and monitoring with FPMON, this should not affect the performance of a website more than the time it takes to load a website and its third-parties. The delay caused by connecting to a service, such as a website, is usually greater than the actual impact of the FPMON on performance and remains therefore unnoticed in all our testing. We further evaluated that FPMON does not impact the performance of a website and its functionality, which is one of the main criteria for FPMON being applicable without detection. Without further interaction, FPMON remains silent and only the icon of a colored fingerprint with an ID text is displayed. When the user clicks on this extension icon, an extension popup appears and a detailed view shows the processed metrics of the current fingerprint activity.

[2] `https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver/MutationObserver`



(A) Index page of fingerprint2.js (B) Landing page of amazon.com

FIGURE 4.4: Clickable pop-up on the extension user interface of FPMON that show sensitive and aggressive features and statistics about the present fingerprinting activity.

At the top, as shown in Figure 4.4, the pop-up shows the current domain and subdomain of a website. At the bottom, we give an overview of how many features and categories we track and which ones we recognize. We also highlight the coverage of the entities and categories found versus the maximum number we monitor. We also classify the categories into two groups, namely sensitive and aggressive features (defined in Section 3.1), and list the detected ones for the user. Newly detected fingerprinting attributes and functions are added in realtime and in order to subsequently expand the list of categories.

4.3 Web-Crawler FPCRAWL

In order to analyze the metrics generated by the FPMON extension, as described in Section 4.2, we need to find a way to automate website visits and collect monitoring data on a larger scale across a variety of websites. Typically, crawling on the Web is done through a series of REST POST or GET requests to extract data from the website or communicate with services on different servers. In this scenario, the client sends a request to the service, and the service responds accordingly. However, this approach is unsuitable for our purpose because we need an active DOM with an active JavaScript engine to perform our measurements at runtime. In addition, many websites use bot detection services, like evaluated in [44], which check requests to see if they were sent by a known automated service. Once a bot is detected, the site may change the functionality or be unreachable to packets from the bot's originating IP. However, these limitations do not seem to affect the instrumentation and automation of a fully functional web browser itself.

4.3.1 Browser Instrumentation

Our goal with FPCRAWL is to instrument and imitate user-like behavior and discover the different functionalities of different websites, as a normal user would do. The instrumentation of a browser can be done in different ways, which can differ in programming and scripting languages and instrumentation levels. For our purpose, the programming language Python3 and the Python library Selenium are best suited. Both technologies are very well documented and are used for all types of crawling and tasks that require web automation. This is great for developing relatively error-prone and easily debuggable software. Selenium is a powerful package that provides a way to communicate with a browser at both the process level and the JavaScript and user interface level. In our architecture, Selenium handles a multitude of tasks in the crawling environment.

- Manage the Chromium web browser process.
- Load the custom extension FPMON.
- Visit specific URL's.
- Check for a FPMON success HTML Element.
- Surveillance of different timeouts on process, application, JavaScript level.

In combination with our extension FPMON, we monitor the behavior of websites and the browser itself. We take momentary records of the monitored data on page completion, timeout, or any other event, as explained in Section 4.2. We then send the data to our listening HTTP server and store it for further analysis. We developed a technique to signal Selenium that the FPMON extension has completed its job successfully its job.

```
  // Create a success element to signal Selenium and the crawler
  // that everything worked as expected.
  var myDiv = document.createElement('div');
  myDiv.id = 'fpmon_success';
  document.body.appendChild(myDiv);
}
```

FIGURE 4.5: FPMON sending metrics to the HTTP server and signaling that the task completed successfully.

Locating the FPMON success element in the current DOM of a browser indicates that the metrics of the FPMON extension have been calculated successfully and the process may be terminated sooner. We also use Selenium to quickly uncover non-functional sites. We can determine if there are problems related to either JavaScript or the browser. We detect errors, such as non-resolving DNS, incorrect use of the REST protocol, e.g. HTTP, HTTPS, and any JavaScript-related error that could render our extension unusable.

The accurate detection and handling of errors dramatically improves the performance of FPCRAWL. For example, the correct handling of errors, such

as DNS or 404 errors, is especially useful since, without these errors, a timeout exception would always be thrown at the expense of a full timeout duration. For timeouts in FPMON and FPCRAWL, we choose the following setup.

- **Initial 20 seconds:**
  FPMON runs from startup and monitors until at least 20 seconds after the full page load event of a website has been triggered to cover all scripts and functionality that depend on this event.
- **Timeout after 30 seconds**:
  FPMON waits a maximum of 30 seconds for a page load complete event to occur. If it does not reach this event, the process is aborted and retried.
- **Chromium Webdriver 45 seconds**:
  If the Chromium process and FPMON monitoring takes longer than 45 seconds, this is counted as a timeout.
- **FPCRAWL Process Selenium 60 seconds**:
  When a Docker browser node is active for more than 60 seconds, it will be terminated or restarted.

We have set the internal FPMON timeout to 20 seconds, as this is a trade-off between the completeness of the data and the time spent browsing the web, where 100% is hardly achievable because of the complexity of possible errors in the Web (e.g. page not resolving, bad connection). The instrumented browser will stay on a web page for 20 seconds after the page-complete event up to a maximum of 30 seconds to generate metrics from fingerprint activity. Otherwise, if a page is not fully loaded, send a timeout notification to the HTTP server after 30 seconds to indicate an error and missing monitoring data. Selenium tracks the instrumented web browser nodes. To avoid errors, Selenium shuts down the web driver instance and browser after 45 seconds to prevent deadlocks. In addition, the crawling process itself is terminated and restarted after 60 seconds if the Selenium process does not return as expected.

On the backend of our crawler runs an HTTP server that collects POST requests sent by the orchestrated web browser. The server listens on a port on the localhost, waits, and collects all requests sent by the FPMON extensions on all instrumented browsers. Additionally, the HTTP server is the main data management component of the crawler. For each URL visited by Selenium, the server logs each (valid data or timeout) output of FPMON and FPCRAWL. The received data is then parsed and stored in a human-readable CSV file that is saved for further analysis.

4.3.2 Orchestration

For the orchestration of larger scans required for our analysis, discussed in Chapter 5, we choose to manage the scans with Docker. This gives us insight into what happens in case of failures on a single node, while giving us the

performance and time advantage to scale our scans to larger server machines. Our Docker architecture consists of multiple nodes that interact with each other.

- **Selenium Hub**
  The Selenium Hub is the component that manages and distributes jobs to the available browser nodes.
- **Chrome Browser Node(s)**
  The Chrome nodes are the actual browsers that are pre-installed with the FPMON extension and monitor fingerprint activity.
- **HTTP Server**
  The HTTP server collects the requests along with the fingerprint metrics sent by each browser node after visiting the specified website.
- **Crawler Node**
  The crawler node manages the job queue and sends new tasks to the hub for distribution.

The biggest challenge here is to correctly configure the de- and registration events from browser nodes to the Selenium hub in case of errors while they are loaded by other nodes. We have found that one way to keep the monitoring data consistent is to restart the hub and browser nodes in case of an error event, which otherwise could lead to a deadlock and delay all jobs until the system is restored. Restarting the nodes affects performance but makes the setup capable of scanning huge lists of consistent output data.

Chapter 5

Evaluation

In this chapter, we present the evaluation part of this work and show the results of our analysis. We present three studies that apply the technologies we have developed, which are explained in Chapter 4. In addition, we discuss and evaluate the results and highlight any interesting artifacts that we encountered in the subsequent studies.

5.1 Study 1: Real-World Hunt for Fingerprints

In order to obtain a broad overview of the behavior of fingerprint services used on the Internet, our first study aims to examine a large number of particularly interesting websites. With our tools FPMON and FPCRAWL we are able to investigate whether and to what extent a website uses known fingerprinting techniques. In this analysis, we select 45 known websites, ranging from, e.g. government sites to whistle-blowing, e-commerce, and news sites to cover a wide range of topics and monitor their applied fingerprinting activity. With this analysis, we aim to gain further insight into the following three different fingerprint and privacy related tasks.

1. Locating aggressive fingerprinting websites.
2. Insight into the distribution of sensitive and aggressive hidden fingerprinting techniques used.
3. Measuring the impact of the different privacy enhancing features and browser extensions:
    - Adblock [83]
    - EFF Privacy Badger Tool [77]
    - DuckDuckGo Privacy Essentials [84]
    - Firefox strict-mode [85]

5.1.1 Methodology

To generate the data for further analysis, we first use our tool FPCRAWL to scan the landing pages of the mentioned websites. We scan 45 websites with a minimum of 20 seconds on each page after loading the page to monitor fingerprint activity. A scan takes about 3 minutes on 14 processor cores. For the Docker browser nodes, as described in Chapter 4.3.2, we use Google Chrome

for scanning with default settings and the FPMON extension. We repeat the same scan with an additional extension from the above list to create a unique record for each scenario. We will also perform two additional scans, using first a clean Firefox node and second, a Firefox node with the privacy setting *strict-mode* enabled. strict-mode is a unique configuration setting of the Firefox browser that improves privacy by disabling social, cross-site, cryptomining, and fingerprint tracking. As a result of finding some artifacts, as described in Section 5.1.2, we decided to perform another scan manually with a clean Chrome browser. For this particular scan, we load our FPMON extension into Chrome, as we did with the first scan, and manually accept any cookie banner that appears. The goal for all scans is to see the effectiveness of the different configurations or extensions compared to the original clean Chrome scan.

For the second task, we collect all links from a website belonging to the same domain. With this, we generate a comprehensive list of directories and endpoints of a website to monitor differences in fingerprinting behavior. The script generates about 3.500 unique entries from an input list of 45 domains with a link depth of one and about 28.000 unique entries with a link depth of two. For the sum of eight scans, the following three monitored metrics are of particular interest for this and the following studies.

- The **calls for fingerprinting and categories** tells us what fingerprinting activity takes place on this site. At the same time, it shows the aggressiveness and general behavior of the fingerprint.
- The **number of calls and categories** marks the target information of the fingerprint and outlines possible multiple uses of the fingerprint.
- The **sequence of calls and categories** that occur allows us to generate a signature of the fingerprint behavior and is relevant to find similarities between the fingerprints used on each site.

We performed all scans several times and were able to achieve almost identical results with only minimal differences. However, the occurring differences are negligible and most likely caused by the dynamic loading of the page and have no influence on the functionality of a website or our measurements with the FPMON extension.

5.1.2 Results

In this section, we present the results of our analysis with the extension FPMON and the crawler FPCRAWL and examine the expected and unexpected behavior when taking fingerprints. All data in the images shown in this section are measured sequentially, to increase the accuracy of our generated data, and to reduce the noise caused by possible changes in the fingerprinting behavior of the website. We perform all scans multiple times in the docker crawl environment to validate the measured data and we have also manually tested and verified artifacts, errors, and unexpected results in a non-dockerized and typical browser environment that a user would experience.

In the following Table 5.1, we show the results of our base scan with a clean Chrome browser with only the FPMON extension loaded. We use the data from this particular scan to further compare the data with our other scans, which cover different settings and configurations since this is a scan with default configurations. In Table 5.1, we display the fingerprinting behavior of 25 of the 45 most relevant sites for us. The websites cover a wide range of content topics and are sorted in decreasing order according to their fingerprint score, which is simply derived from the coverage of a website's fingerprint category. The metric score represents the activity of a fingerprint and is used in this study to compare different configurations and their impact on the fingerprint. In addition to the measured coverage of the fingerprint categories, we show the *Entity Coverage*, which indicates how many fingerprint views we have accurately identified, and the *Aggressive Coverage*, which shows the proportion of measured aggressive categories. We also classify the list into four sections (horizontal lines), which we define high, medium, low, and no fingerprint activity of a website.

Domain	Content Topic	Entity Coverage	Category Coverage	Aggressive Coverage	Score
metacafe.com	Video Sharing	93/116	38/40	17/18	95%
easyjet.com	Flight Service	62/116	29/40	12/18	73%
nasdaq.com	Stock Market	52/116	28/40	11/18	70%
bankofamerica.com	Finance	57/116	26/40	8/18	65%
savethechildren.org	Non-profit	74/116	25/40	12/18	63%
nytimes.com	News	49/116	24/40	12/18	60%
coinbase.com	Privacy Service	68/116	23/40	11/18	58%
alibaba.com	E-commerce	65/116	22/40	10/18	56%
healthcare.gov	Healthcare	45/116	22/40	11/18	56%
fingerprintjs.com	Fingerprinter	61/116	21/40	11/18	53%
vyprvpn.com	Privacy Service	64/116	19/40	6/18	48%
google.com	Web Browsing	22/116	13/40	7/18	33%
theguardian.com	News	29/116	12/40	3/18	30%
paypal.com	Finance	19/116	9/40	1/18	23%
pornhub.com	Pornography	19/116	9/40	2/18	23%
youtube.com	Video Sharing	26/116	7/40	3/18	18%
nsa.gov	Intelligence Service	11/116	6/40	2/18	15%
archive.org	Digital Library	7/116	5/40	0/18	13%
wikipedia.org	Encyclopedia	12/116	5/40	0/18	13%
duckduckgo.com	Web Browsing	10/116	4/40	0/18	10%
netflix.com	Streaming Service	15/116	4/40	0/18	10%
addthis.com	Fingerprinter	7/116	3/40	1/18	8%
torproject.org	Privacy Service	4/116	1/40	0/18	3%
thepiratebay.org	File Sharing	0/116	0/40	0/18	0%
wikileaks.org	Whistleblowing	0/116	0/40	0/18	0%

TABLE 5.1: Several websites use large portions of the spectrum of available fingerprint information measured with FPMON on a clean Chrome browser.

Since the category coverage and aggressive coverage are related to each other, we are able to classify values based on the scores as follows.

- **High** Score of 30% or more
- **Medium** Score in between 15-30%
- **Low** Score in between 15-5%
- **None** Score smaller than 5%

We consider fingerprint activity to be *high* if there is medium or high(>10) category coverage and multiple(>2) aggressive fingerprint categories are detected. *Medium* fingerprint activity occurs when there is significant(6-10) coverage of fingerprint categories, but very few(<=3) to none of them are considered aggressive. *Low* fingerprint activity occurs when there is a small(<=5) number of fingerprint categories and no aggressive category is detected and *none* for no fingerprint activity.

The most aggressive fingerprint activity we measured in this scan on landing pages is the video-sharing platform *metacafe.com*. With a coverage of 38 out of a maximum of 40 recognized fingerprint categories, the domain reaches a result of 95% and, as we found out in the further analysis in Studies 2 and 3, is one of the five areas with the same result of 95% that we measured in this thesis. With 93 out of 116 detected entities and 17 out of 18 detected aggressive fingerprint categories, this area comes close to using the full spectrum of available fingerprints we know of and we are able to detect. We found nine domains in this scan with the same or higher aggressiveness of 53% as the open-source fingerprint service *fingerprintjs.com*, and we rate the comparatively low category coverage of this full fingerprint service as an alarming signal, as we see that there are significantly higher values for other web sites. The domain *google.com* is unique in its behavior and particularly remarkable, because the sites have low coverage of entities(22) and categories(13) compared to all other domains in the *high* classification, but has a comparatively high aggressive coverage of 7. Google most likely only uses fingerprint activity on a small scale but uses it very selectively to extract fingerprint information. In addition, we have measured an unexpectedly low rating of fingerprint activity for *paypal.com* at 23%, *pornhub.com* at 23%, *youtube.com*. 18%, and *netflix.com* with 10% compared to websites of similar size and topic. Expected cases in the *medium* and *low* sections are *archive.org*, *wikipedia.org*, *duckduckgo.com*, *torproject.com*, and *wikileaks.org* whose public relations work is closely linked to discretion and privacy. We have measured that these sites primarily use features that enhance user behavior rather than extract sensitive information. Nevertheless, these features can be used to a certain extent to generate fingerprint data. We estimated, however, that this is unlikely to be the case, as we measure this behavior primarily on sites that are generally known to be more privacy-conscious.

Within the scope of the sites we have selected in this study, we have deliberately chosen those sites, that would be controversial if any fingerprint activity would be found. The non-profit organization *savethechildren.org*, privacy enhancing services, such as *coinbase.com* and *vyprvpn.com*, or public healthcare providers, such as *healthcare.gov* have unexpectedly high fingerprint activity and are part of the discussion in Section 5.4.

In the second scan of this study, we investigate the effects of a clicked cookie banner and its active manipulation on fingerprint activity. To do this, we re-run the scan we previously performed with the clean Chrome browser and click manually on the cookie banner. Table 5.2 shows the comparison between the coverage of the detected fingerprint categories and the score that represents the aggressiveness over the range of 45 websites sorted in descending order by score.

Domain	Content Topic	Cookie Banner Deactivated		Cookie Banner Activated		Score Difference
nasdaq.com	Stock Market	28/40	70%	12/40	30%	- 40%
healthcare.gov	Healthcare	22/40	56%	21/40	53%	- 3%
medicare.gov	Healthcare	22/40	56%	21/40	53%	- 3%
vyprvpn.com	Privacy Service	19/40	48%	10/40	25%	- 23%
sba.gov	Government	16/40	40%	12/40	30%	- 10%
twitch.tv	Streaming	16/40	40%	14/40	35%	- 5%
reddit.com	Social Media	13/40	33%	9/40	23%	- 10%
spiegel.de	News	11/40	28%	18/40	45%	+ 17%
paypal.com	Finance	9/40	23%	10/40	25%	+ 2%
addthis.com	Fingerprinter	3/40	8%	34/40	85%	+ 77%
yahoo.com	Web Browsing	3/40	8%	9/40	23%	+ 15%
panopti-click.eff.org	Fingerprinter	1/40	3%	21/40	53%	+ 50%

TABLE 5.2: Fingerprint activity measured with FPMON on a clean Chrome browser with cookie banner clicked shows increases and decreases in the score.

The fingerprint provider *addthis.com* shows the most significant effect on a clicked cookie banner. Accepting its cookies and privacy policy activates the fingerprinting behavior, which increases from a very low 8% to a very aggressive 85%. The second fingerprint service in this list, *panopticlick.eff.org*, shows a 50% increase in fingerprint activity - from 3% to 53%. However, this action is not triggered by a cookie banner but is activated by a "Test Me" button. It is only a sample with this action we added because of its completeness and comparability using a fingerprint service. Nevertheless, significant increases from 8% to 23% for *yahoo.com* and 28% to 45% for *spiegel.com* indicate that this is a common use to trigger additional fingerprint activity. However, clicking on a cookie banner not only shows an increase in the number of fingerprint categories recognized but also can result in a decrease of categories. The stock site *nasdaq.com* fell the most from 70% to 30%, while other sites, such as *healtcare.gov*, *vyprvpn.com*, *sba.gov*, *twitch.com*, *reddit.com*, and *paypal.com* fell in the range of 2-23%. The measured activity drops occurred mainly when reloading the page where the cookie banner was removed.

To our knowledge, this behavior can be used to shift the fingerprinting capability into the context of the underlying functionality of a website. Alternatively, as in the cases where we measure increased fingerprint activity, it allows for direct and aggressive fingerprinting by accepting the privacy policy of that site. If no cookie is set and no banner is clicked, sites most likely choose to fingerprint their visitors directly. If a cookie is set, they can

do this later and more covertly after login or session management. In this case, the activity is difficult to measure and evaluate, as the scope changes from a single website to a session and thus dependent on the validity of a cookie. To address data protection concerns, the GDPR requires websites under EU law to present their data collection and distribution in a transparent manner. However, the data we have measured indicate, that even if websites are compliant with the GDPR by having a cookie banner and displaying their privacy policies, they are unlikely to accurately present what and how they are tracking and, in addition, aggressive data collection measures are not listed.

Our next scan provides some insights into the ways that a web browser user can improve their privacy on the Internet with different browser extensions and configurations. To examine differences in privacy improvements and to further measure the impact of these improvements compared to our previous clean scan, we will scan the list of 45 different pages with the configuration loaded. Table 5.3 shows a selected list of the most variations and interesting results we found, with either unexpectedly high or unexpectedly low impact. The table is sorted by the score we measured in the first clean Chrome scan in Table 5.1 and is compared to the three different scores of the scans with the extensions *Adblock*, *Duckduckgo privacy Extension*, *Privacy Badger*.

Domain	Content Topic	Score Chrome	Ad-block	Duck-duckgo	Privacy Badger
metacafe.com	Video Sharing	95%	95%	95%	20%
easyjet.com	Flight Service	73%	73%	73%	73%
nasdaq.com	Stock Market	70%	70%	70%	68%
bankofamerica.com	Finance	65%	58%	65%	65%
savethechildren.org	Non-profit	63%	45%	23%	43%
nytimes.com	News	60%	58%	18%	60%
coinbase.com	Privacy Service	58%	58%	58%	58%
fingerprintjs.com	Fingerprinter	53%	53%	53%	53%
sba.gov	Government	40%	25%	10%	18%
theguardian.com	News	30%	23%	15%	18%

TABLE 5.3: Privacy enhancing browser extensions have a positive impact on limiting fingerprint activity, in contrast to no extensions used. Measured with FPMON on a Chrome browser with one extension loaded at a time.

Privacy Badger has the most noticeable effect on *metacafe.com*, where the value of 95% is reduced by 75% to 20%, while Adblock and Duckduckgo do not restrict fingerprint activity at all in this case. We found another significant impact with the Duckduckgo extension on *nytimes.com*. Here the extension reduced the score from 60% to 18%, while Adblock decreased the activity by a negligible 2% and Duckduckgo by 0%. An example of a website where all three extensions effectively limit fingerprint activity is *sba.gov*. The score of 40% measured with the clean Chrome browser is reduced by 15% with Adblock, by 30% with Duckduckgo and by 22% with the Privacy Badger extension. Unexpectedly, one site where no extension restricts anything

is the fingerprint site *fingerprintjs.com*. In this case, all extensions have the same score of 53% as the clean Chrome browser. Other websites where no restrictions apply are *easyjet.com* and *coinbase.com*. However, these sites show fingerprint activity that is restricted on other sites. This indicates that the normal behavior of any extension is most likely dependent on blacklisting. This means that an extension that has a particular fingerprint script or a third party script in their blacklist will be blocked. If the tracker is not on the list, no restrictions will take place and the fingerprint will collect data as if no privacy extension was loaded. However, the table shows that each extension has an impact on at least some of the monitored websites. The data we collect is unexpectedly consistent in its reproducibility but inconsistent in its coverage of fingerprint functionality across multiple sites. Therefore, the effectiveness of an extension depends on the website itself and its functionality. Since this part of the study is about examining our monitoring capabilities with the FP-MON, we do not evaluate whether one of the extensions in question is better or worse than the other. On the contrary, we have found evidence that there is a measurable and positive effect on all three privacy-enhancing extensions.

The following result shows a different setting with measurements made with the Firefox browser. In this case, we are evaluating an internal browser functionality *strict-mode* that improves the privacy of its users. Table 5.4 shows the comparison of the detected evaluation and fingerprint categories of a standard Firefox browser on the left side and a standard Firefox browser in strict-mode on the right side. The websites listed in this table represent the subset of the most diverse results from the list of 45 websites we scanned.

		Firefox		Firefox Strict	
Domain	**Content Topic**	**Categories**	**Score**	**Categories**	**Score**
metacafe.com	Video Sharing	38/40	95%	7/40	18%
easyjet.com	Flight Service	29/40	73%	21/40	53%
nasdaq.com	Stock Market	28/40	70%	28/40	70%
nytimes.com	News	24/40	60%	24/40	60%
coinbase.com	Privacy Service	23/40	58%	23/40	58%
alibaba.com	E-commerce	22/40	56%	22/40	56%
healthcare.gov	Healthcare	22/40	56%	21/40	53%
medicare.gov	Healthcare	22/40	56%	21/40	53%
fingerprintjs.com	Fingerprinter	21/40	53%	21/40	53%
airbus.com	Flight Service	17/40	43%	16/40	40%
sba.gov	Government	16/40	40%	15/40	38%
ny.gov	Government	15/40	38%	15/40	38%
breitbart.com	News	38/40	95%	7/40	18%
google.com	Web Browsing	10/40	25%	10/40	25%
amazon.de	E-commerce	8/40	20%	8/40	20%
pornhub.com	Pornography	9/40	23%	7/40	18%
stackoverflow.com	Knowledge Market	8/40	20%	4/40	10%
europarl.europa.eu	Government	5/40	13%	3/40	8%
addthis.com	Fingerprinter	34/40	85%	7/40	18%
panopticlick.eff.org	Fingerprinter	21/40	53%	21/40	53%

TABLE 5.4: Privacy enhancing browser setting Firefox strict-mode has a slight, yet positive impact on limiting fingerprint activity, in contrast to default settings.

The first thing that stands out is that the Firefox browser has almost identical results in terms of scores and fingerprint categories detected, compared to the scans we previously performed with the Chrome browser. This shows that the fingerprint behavior is website dependent and, therefore, not browser specific at all. Firefox strict-mode shows a similar overall impact on websites fingerprinting activities. Like the Privacy Badger extension of Table 5.3 the Firefox strict-mode reduced the activity of *metacafe.com* by 77% (whereas Privacy Badger has 75%), while the inconsistently triggered fingerprint of *breitbart.com* was reduced from 95% to 18%. However, the Firefox strict-mode has limited *easyjet.com* by 30%, but has not reduced the coverage of the fingerprint categories *nasdaq.com*, *coinbase.com* and *nytimes.com*. The low category coverage, but aggressive fingerprint actions of *google.com* remain unaffected by the privacy setting Firefox strict-mode, as do the fingerprinters *fingerprintjs.com* and *addthis.com*, where no effects of a restriction on fingerprint behavior can be measured. With the Firefox browser and its privacy-enhancing configuration strict-mode, we measure the same consistency in reproducibility along with the similar inconsistency within the restrictions on different websites, as seen in the previous Chrome scans. This also indicates a restriction behavior that depends more on blacklisting known trackers than on preventing the generation of sensitive information on all websites.

To verify our measurements, which show that mainly blacklisting techniques are used, we further investigate the distribution of blocked fingerprint categories in the clean Firefox and Firefox strict-mode scan data. The expected result here is to see groups of categories blocked by blacklisting all together, thus highlighting categories that are not part of the blacklisting approach.

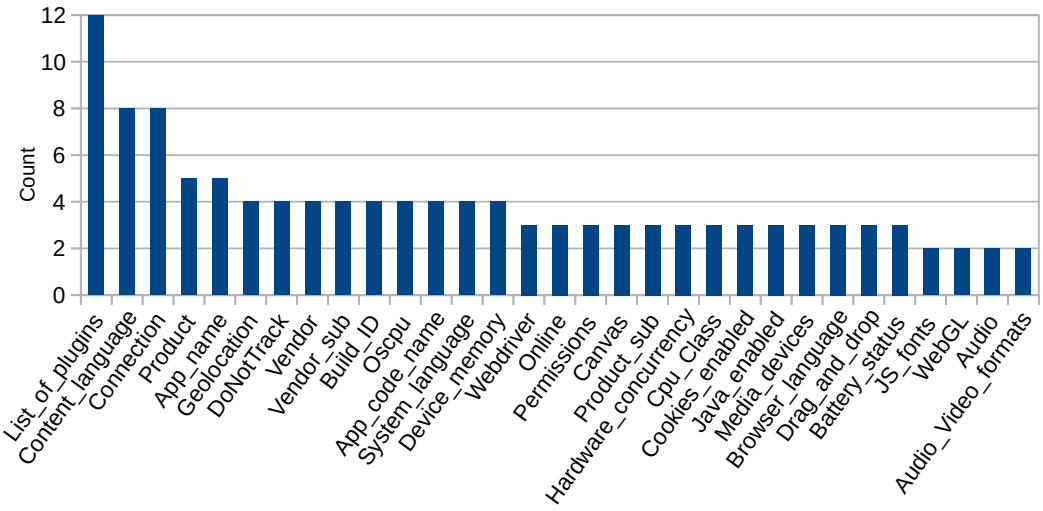


FIGURE 5.1: Only a small number of individual categories are blocked by Firefox's strict-mode, but all are blocked if a tracker can be detected as blacklisted.

Figure 5.1 shows the differences in the detected fingerprint categories we measured on the list of 45 websites. In this analysis, we calculate the difference in the number of recognized fingerprints on each site where we see an impact on fingerprint activity. Within the 40 categories covered by the FP-MON, we monitor 31 different fingerprint categories, which are restricted by Firefox in the strict-mode configuration. The most restricted category is *List_of_Plugins*, whose execution stop we measure at 12 of 45 sites. The second most common restrictions are *Content_language* and *Connection* with a number of 8, followed by *Product* and *App_name* with an occurrence of 5. However, this restriction behavior is unexpected, as these categories are used primarily for website functionality. Although we classify *List_of_plugins* as an aggressive fingerprint category, it is unlikely that preventing a single instance of this functionality will have a positive impact on a site's fingerprinting or privacy practices. Similar to the *UserAgent* category, which is not present here, the categories are mostly used to provide well-functioning and dynamic website functionality, although they provide some fingerprinting capabilities too. Further to notice is that there are three plateaus with counts four, three, and two of fingerprinting activities visible. Knowing that the Firefox scan restricted 4 full but different fingerprinters on *addthis.com*, *breitbart.com*, *easyjet.com*, and *metacafe.com* explains the wide coverage of this difference analysis. If we now subtract the four blacklisting approaches from the data available to us, we see that only the five categories already analyzed, List_ of_plugins, Content_language, Connection, Product, and App_name, remain. This indicates that in strict-mode, only minor privacy activities are performed, apart from blacklisting known trackers. Any aggressive fingerprint categories, such as *WebGL*, *JS_Fonts*, or *Canvas*, that are not part of a blacklisted script, are still active.
If we now subtract the four blacklisting approaches from the data we have left, we can see that there are only the five already analyzed categories List_ of_plugins, Content_language, Connection, Product and App_name left. This gives evidence that strict-mode implements just minor privacy protection activities apart from blacklisting known trackers and any aggressive fingerprinting categories like *WebGL*, *JS_fonts* or *Canvas* not part of a blacklisted script are still active.

In the following part of this study, we examine the coverage of the fingerprint categories in the areas that we have previously scanned and analyzed on a larger scale. The goal here is to find the maximum coverage of a particular domain where all subdomains fall within the scope since scanning landing pages is not the best way to make a statement about the fingerprint activities of a website. We generate an entry list for our crawler FPCRAWL from the previous 45 websites with a link depth of two, resulting in about 28.000 unique links to websites from the same domain. Table 5.5 shows the number of fingerprint categories we found on the landing pages from Table 5.1 and compares it to the now measured maximum category coverage we find with this scan.

Domain	Landing Page	Domain Max	Max fingerprint activity measured on
breitbart.com	13	38	www.breitbart.com/[...]
archive.org	5	37	archive.org/donate/
thepiratebay.org	5	31	thepiratebay.org/browse.php
ebay.com	11	28	signin.ebay.com/[...]
bankofamerica.com	26	27	www.bankofamerica.com/credit-cards/[...]
pastebin.com	4	22	deals.pastebin.com/
sba.gov	16	20	www.sba.gov/
paypal.com	9	18	www.paypal.com/am/welcome/signup[...]
amazon.de	9	16	www.amazon.de/gp/help/customer/[...]
stackoverflow.com	8	15	ru.stackoverflow.com/
pornhub.com	9	14	www.pornhub.com/view_video.php?[...]
spiegel.de	11	13	www.spiegel.de/newsletter
youtube.com	7	12	www.youtube.com/gaming
nsa.gov	6	10	www.nsa.gov/about-us/[...]
wikileaks.org	0	4	our.wikileaks.org/Main_Page

TABLE 5.5: The number of recognised categories increases significantly if we extend the scope from the landing page to all subpages of a website with a link depth of two.

This dataset reveals an unexpected result and provides examples of how different companies use fingerprints in the wild. For example, at *archive.com* we found low fingerprint activity on most of their sites. However, the website *archive.com/donate/*, where users can make monetary donations, has some of the most aggressive fingerprint activity on all of the scans and websites we measured. This behavior also indicates that highly targeted usage behavior by companies that collect information from specific users is accurately applied. This scan also shows a dramatic increase in fingerprint activity on login or log-in pages, such as on *ebay.com*, which increases the categories detected from 11 to 28, and *paypal.com*, where 9 to 18 detected fingerprint categories increase. On these pages, the fingerprint is likely to be sent directly via the first request without being stored via the browser storage API before. Another interesting finding is that we measure differences in fingerprint activity caused by specific content, such as certain languages, as seen with *ru.stackoverflow.com*, the gaming section in *youtube.com/gaming/*, selected videos on *pornhub.com/view_video.php?* or the newsletter page on *spiegel.de/newsletter*. In addition, subdomains such as *deals.pastebin.com* may have different fingerprint activity and structure of a website than the domain's landing page. However, our scans give no indication, as to whether this behavior is intentional or whether small to medium variations in fingerprint activity that we measure could also be caused by dynamic advertising or page loading.

In this study, we further found some unique behavior on the *addthis.com* domain, where a fingerprint is immediately activated when we click on the cookie banner. The behavior, where accepting a website's privacy policy results in increased fingerprint activity is typical behavior, as most websites recognize this action and perform their fingerprint activities on the next page load. However, this is not the case for *addthis.com*. At this domain, a fingerprint script remains silent and waits in the background to be executed.

Clicking the cookie banner is the active trigger to start the execution of the fingerprinting script on this domain. In contrast to the artifact mentioned above, we observed a similar behavior of the file-sharing platform *thepiratebay.org.com*. Only this area applies fingerprinting techniques when a user is unknown to the site and a cookie set by this site effectively disables fingerprinting activity. We did not expect that this would be the case for most fingerprints, but as can be seen in Table 5.2, a large percentage of the fingerprints measured behave this way. Another anomaly is the fingerprinting behavior of the *breitbart.com* domain. We have extensively tested the possible triggers for this very aggressive fingerprint to always intercept the fingerprint while our scans are running. As far as our tests found out, the fingerprint on this domain acts randomly, regardless of whether a user is visiting a landing page or a subpage. Fortunately, this does impact the quality of our measurements, since we detect the full scope of actions once we gathered all behavioral data to perform our analysis.

5.2 Study 2: Fingerprinting Fingerprinters

For the second study of our research about web-device fingerprinting, we investigate the distribution and activity on a much larger scale. In this analysis, we aim to generate data about which fingerprinting technique and categories are currently in use and how widespread fingerprinting techniques are applied.

5.2.1 Methodology

Since our FPCRAWL crawler is capable of scanning larger lists, we decided to scan the landing pages of the 10.000 most visited websites for this analysis. The Alexa 10k list [86] contains the 10.000 most popular websites on the internet regarding their caused traffic and is usually used for comparison in the current research [23, 52]. In this study, we perform our scan with a clean Google Chrome node in the docker crawl environment and only preload our FPMON extension to monitor fingerprint activity. This scan takes approximately 7 hours on 14 processor cores. As described in the previous study, in Section 5.1.1, metrics, such as occurrence, number, and order of fingerprint categories, and calls are of particular interest for analysis. As in the first study, we performed the scans several times and were able to obtain almost identical results with minimal differences. However, these differences are negligible and most likely caused by the dynamic loading of pages.

5.2.2 Results

In this section, we present the results of the second study with FPMON and FPCRAWL. The data we obtained from the scan of the Alexa 10.000 list shows some additional metrics compared to the scans we performed in Study 1.

Of the 10.000 sites we scanned, we were able to generate a valid measurement on 88% of the sites. The loss of the remaining 12% of the data is caused either by server-side timeouts, websites that do not resolve, or critical errors that prevent our FPMON extension from running correctly. 98% of the valid data we measure reaches the "JavaScript page load complete" event, which implies a successful measurement without errors from our side. A timeout of a maximum of 30 seconds residence time will, in case of errors, cause the scanning of the affected page to be aborted, and the measured fingerprint activity will still be captured up to this point. Failure to achieve this event can have various causes and, to our knowledge, is not caused by the FPMON itself in almost all incidents. However, the fingerprint information generated by the missing 2% is still valid data, as any JavaScript fingerprint activity is still present even when a page is fully loaded. Across all scanned websites, we measured an average page load time of 5.95 seconds, including the 2% that reached a 30 second timeout.

The following Figure 5.2 shows the distribution of how many websites actively use fingerprinting techniques. This includes the frequency of fingerprint categories that are captured by our large-scale scan. The first noticeable thing is the relatively large impact when no fingerprint categories are captured.

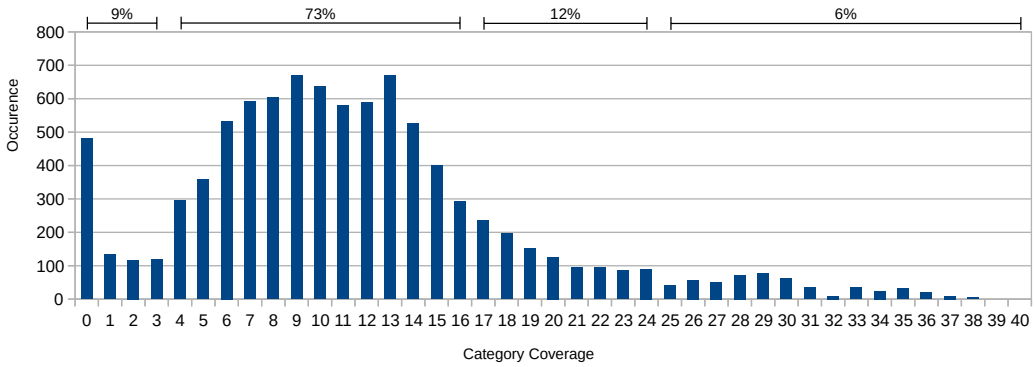


FIGURE 5.2: The majority of websites scanned with the FPMON extension have a range of 4 to 38 different categories and show that fingerprinting has become very popular on the Internet.

The reason for this could be that JavaScript is not used on these 481 websites. In addition, we find that an equally small number of websites have a detection rate of less than five fingerprint categories, which together account for 9% of the detection volume. 73% of the websites we measure have fingerprint activity ranging from 4 to 16 categories. For the remaining 18% of websites, we measure activity for which more than 17 categories have been identified. This list of sites can be considered the most aggressive group, while the 6% that have detected more than 25 categories are very aggressive towards user fingerprinting. It is also noteworthy that the coverage values of 29 and 35 are local extremes. This could be the reason for two different, very aggressive

fingerprint scripts, spread across several websites, all using the same fingerprint templates and techniques with only minor differences. We suspect that there are not too many different fingerprint scripts on the market, and we investigate this in Study 3 in Section 5.3.

In Figure 5.3 we show the distribution of the measured fingerprint categories. The five most commonly used categories are *User_agent*. (88%), *Screen_window* (86%), *Content_language* (79%), *List_of_Plugins* (78%), and *Storage* (77%). We observe that most of the aggressively categorized categories can be found on less than 2000 websites. It can be seen that there is a clear difference in the distribution of the aggressive and sensitive fingerprint categories.

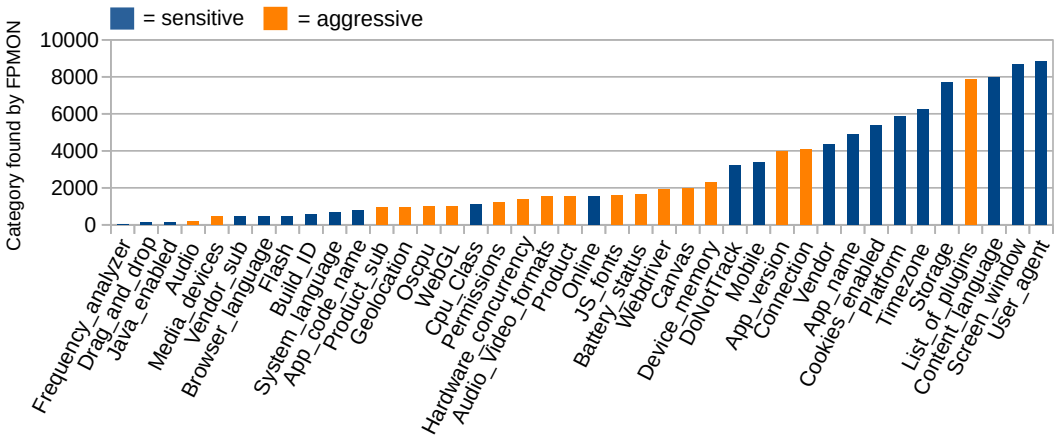


FIGURE 5.3: The most commonly recognized fingerprint categories of all websites scanned by FPMON are sensitive and often required for website functionality. However, when used in fingerprinting, they are often paired with several sensitive or aggressive categories, which can be a threat to privacy.

Since fingerprint scripts cover a wide range of different functionalities in the browser, multiple fingerprint categories are usually covered simultaneously, as general browser functionalities used not for fingerprint capture. Although in some cases, they can be used for fingerprinting, it is not very likely that they will be used as stand-alone fingerprint metrics. The higher the number of categories recognized by the FPMON extension, the more likely it is that the category is not part of an actual fingerprint script. However, the *User_agent* is an exception, as it is used by almost all websites, but contains high bits of entropy to identify a user. Also, our assessment of which categories are considered aggressive and sensitive is accurate compared to the results of our measurements. It can be seen that almost all aggressive categories are present in the lower count portion of the figure which shows the behavior we expected.

In the next Figure 5.4, we show the distribution of the aggressively classified fingerprint categories. It can be observed that the more fingerprint categories are recognized, the more aggressively classified categories are added.

In addition, the most aggressive fingerprint activity (18% of all websites) occurs when category coverage exceeds 16, as shown in Figure 5.2.

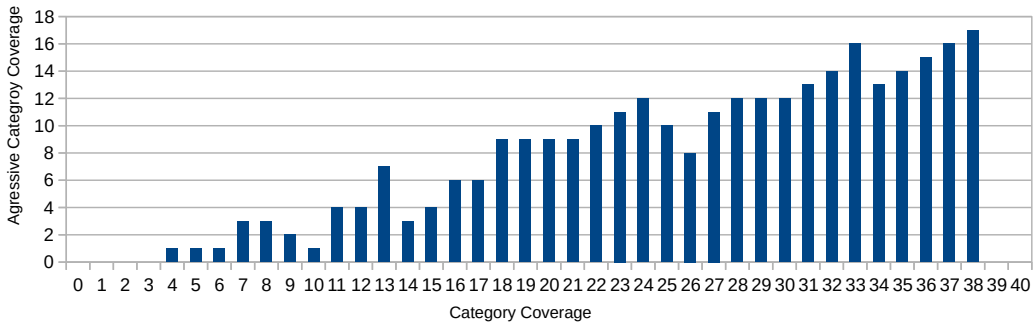


FIGURE 5.4: The more categories the FPMON extension detects on a website, the more aggressive the categories are, with an approximate share of about 50% in all cases.

From this point of view, the coverage value 13 stands out, containing an aggressive share of about 50%. This shows that the medium-aggressive part of the websites scanned by us still contains a high percentage of aggressive fingerprints. It can be concluded that many websites in a relatively low range still use aggressive fingerprint scripts to generate user data. The values in the ranges 0 to 3 and 39 and 40 do not show any activity, as we have not recorded any aggressive fingerprint categories in the lowest range and cannot find websites that cover the entire range of 39 or 40 categories.

5.3 Study 3: Identifying Fingerprinting Networks

In the third study, we analyze fingerprinting behavior and further investigate the data collected in previous studies in Sections 5.1 and 5.2. This study aims to identify networks with the same fingerprinting behavior across the data we collected.

5.3.1 Methodology

The measurements from all scans we have performed in previous studies contain data that we use as a signature for each scanned area. This signature consists of all fingerprint categories that we recognize in real order, and as a result of this feature, it is implementation dependent and unique enough to reliably distinguish fingerprint scripts. The big advantage of this dataset is, that we can see similarities within the signatures that we have already measured and searched for this particular signature or sub-signature in our data set. A signature for a part of a fingerprint script looks like the following (reduced) signatures of canvas and an audio fingerprint

- **"Canvas;Canvas;Canvas;Canvas;Canvas;Canvas;"**
  The sub-signature contains several consecutive canvas category detections because the fingerprint script performs six separate but continuous operations on an HTML canvas element.

- **"List_of_plugins;Audio;Audio;Audio;Frequency_analyzer;"**
  In this example, to create an audio fingerprint, the fingerprinter first checks available plug-ins, such as available audio players, initializes the configuration for a particular audio fingerprinting device, and then sends this data to an HTML frequency analyzer node to create the fingerprint data.

The complete signature of a website is a concatenation of all sub fingerprinting measurements and, in contrast to the two signatures in the single fingerprint category, has enough variation possibilities to be considered unique for a certain fingerprint script. In this analysis, we use a new metric that utilizes the small and large differences in the implementation of different fingerprint scripts to gain insight into their similarity and affiliation. With this available information, we can reliably calculate the Longest Consecutive Subsequence. LCS is a measure to determine the length of the longest consecutive sequence of elements. We use this to generate a fingerprint from a website's fingerprint activity and show whether two websites use different or similar information capture techniques, regardless of differences in their coverage of the fingerprint category.
In this study, we use the data from the list of 45 websites measured in Study 1 with a chrome node (see Section 5.1) and examine the signatures and subsignatures of those in the 10k scan from Study 2 (see Section 5.2). The detection of similar or identical parts of signatures within other signatures strongly indicates that the same JavaScript code is used for fingerprint generation. The higher the LCS score, the more similar are the fingerprint scripts in comparison. The same sequence of calls to generate a fingerprint also indicates that the scripts are identical and may be delivering the fingerprint data to the same fingerprint network.

5.3.2 Results

In this third study to investigate fingerprint activity, we first compared our signature results from the smaller scan in Study 1. We correlate all fingerprint signatures calculated from and to each website included in the list of 45 websites and examine whether we can find the most common signature of a website. We visualize this with the heatmap shown in Figure 5.5 by using the nine most interesting results with the most aggressive and category specific fingerprints we found. The figure shows the calculated LCS score between each pair of websites. The first observation we make is that the overall noise level of similarity from fingerprint activity is in the range of LCS score 5 and 11, and in the case of *breitbart.com* in the range of 7 and 19.

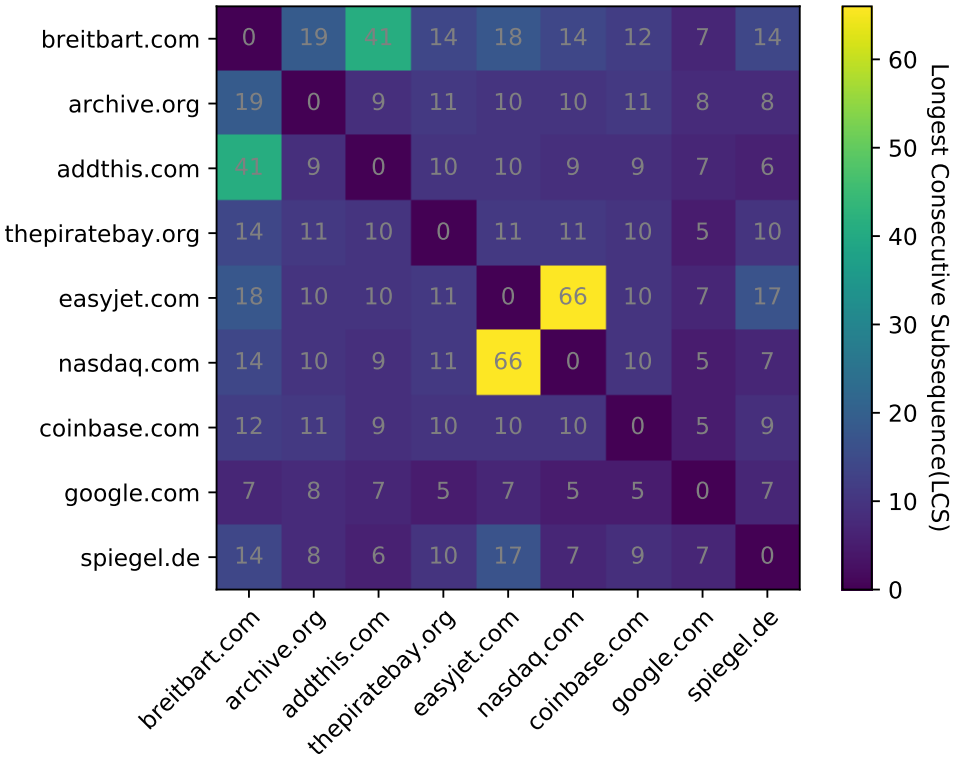


FIGURE 5.5: By correlating the longest consecutive sequence of measured fingerprint signatures, we are able to find two exact matches with identical patterns and numerous identical subsets of not perfectly matching signatures.

This is the result of a fingerprint where almost all categories are used in a different order, but still have a lower signature match, which we consider to be the similarity noise. However, similarity noise comes from the different but technically very similar implementations to perform fingerprint actions in JavaScript. Often, most developers do not modify the syntax of the code too much, as they find the code in the official documentation or on knowledge-sharing websites, in order to solve programming related tasks and problems. Therefore, when performing more complex actions, such as canvas or audio fingerprinting, it is very likely that the syntax and sequence of calls to generate a fingerprint or part of a fingerprint will be similar to those suggested on those websites or in the official documentation. For this reason, all fingerprint signatures have at least a minimal LCS similarity rating. Fortunately, there is a large gap between the noise of low signature matches and the actual signature matches where whole chains of categories are in the correct order and identical category calls are detected. That is the case with the signature of *breitbart.com* with an LCS score of 41 compared to *addthis.com*. Here we calculate a signature match of 41 category calls with the correct order, while the next match with *archive.com* is 19, which can be considered noise since the order of the uniquely recognized categories is completely different. Another signature match is *nasdaq.com* with *easyjet.com*, where we calculate an even higher LCS value of 66. In addition, the domain *google.com*, for example, has

the least similarity of an LCS value of five to eight in its fingerprint signature compared to other websites. With this technique, we can validly distinguish between actual similarity and similarities caused by the use of the same code syntax and categories. It also shows that many different and nearly identical fingerprint scripts are used by websites that we have tested as part of our intentionally small sample list.

Since we can calculate very good results with the LCS-Score and find validated hits for the signatures, we extend the correlation range in the following part of this study. Our goal is now to search for the signatures in the large scan with 10.000 entries, but also for all partial signatures that we have calculated so far. The following Figures 5.6, 5.7 and 5.8 show the score of the longest connected subsequence of ordered entities, which in this case are matches of fingerprint signatures across all 10.000 websites. For reasons of comparability, we further calculate the number of unique categories represented in the signature LCS (*Cat*). The similarities of *spiegel.com* are shown in Figure 5.6a. We found 60 different domains that use a signature that is in the range of LCS values 24 to 39. Each signature within these 60 correlated domains is at present is not only a subset of ordered categories of the *spiegel.com* domain, but is also a subset of the other domains found. Thus *stepstone.de* with an LCS score of 27 is a subset of *spiegel.de* and *bild.de*, *time.de*, *motortalk.de*, all of which have an LCS score of 39. It should be noted that all of the non-noise signature hits shown here are part of or related to the German media, which would indicate that they are part of the same advertising network. The almost perfect result of an entire network consisting of signature hits from only German news sites further supports the evidence and validity of this approach.

Domain	**LCS**	**Cat.**
bild.de	39	5
zeit.de	39	5
motor-talk.de	39	5
stepstone.de	27	5
t-online.de	25	5
wetter.de	25	5
mobile.de	25	5
n-tv.de	25	5
tagesschau.de	25	5
heise.de	25	5
... 50 more	24 – 25	4 – 5

(A) Signature correlation: **spiegel.de**

Domain	**LCS**	**Cat.**
aviasales.ru	93	20
tophotels.ru	91	20
torrentgalaxy.to	91	20
viatorrents.com	91	20
xhamsterlive.com	88	19
pccomponentes.com	88	19
chaturbate.com	88	19
anyporn.com	84	17
vk.com	72	16
coursera.org	70	16
... 140 more	40 – 91	9 – 20

(B) Signature correlation: **coinbase.com**

TABLE 5.6: When correlating fingerprint signatures of 10k websites, we found high LCS values for spiegel.de, which revealed websites that were all related to the German media, and for coinbase.com sites, that showed very high matches with a variety of porn, torrent, social media and e-commerce websites.

On the contrary, as shown in Figure 5.6b, we correlated signature hits for the domain *coinbase.com* and found a wide range of related page topics. The topics range from flight booking pages to torrents and porn sites, e-commerce, social media, and e-learning, and together with the findings from *nasdaq.com* in Figure 5.7b, they are the most diverse set of signatures we have found for a fingerprint network. To us, it is unexpected that these very well-known websites, such as the Russian Facebook derivate *vk.com* or the well-known learning platform *coursera.com* share identical fingerprint signatures with somewhat shadier torrent and porn sites. The property of finding identical subsets as we discovered on *spiegel.com* is not the case for all signatures we correlate on *coinbase.com*, but is evident in this comparison for the highest 101 of the 150 signatures. *Coinbase.com* has a significantly higher LCS score range of 40 to 93 due to the generally higher coverage of the recognized fingerprint categories of 23 compared to the 13 from *spiegel.de* (see Figure 5.1). where more than 65% of the 150 similar signatures discovered have an LCS of more than 70. However, the range of distinct categories found in the most similar hits of signatures varies widely. We can see that *spiegel.de*, with 5 out of a maximum of 12 unique categories found in the signature, has enough variation to differentiate and filter all non-German media pages. This is great because it supports the fact that *coinbase.com* has found 20 of a maximum of 23 unique categories, making it an almost perfectly similar signature match.

In the following Figure 5.7 we show the signature correlation result of the domains *addthis.com* and *nasdaq.com*. As we have seen in Figure 5.5, we found two signature matches in our smaller scan, whose we have located in the correlation with the signatures of 10.000 websites. Both matches *breitbart.com* and *easyjet.com* have a relatively high LCS value of 41 and 66 and a very high category coverage within the LCS signature at 33 and 23, compared to our other finds from the signature correlation.

Domain	LCS	Cat.
globalnews.ca	79	34
express.co.uk	76	32
foxsports.com	76	32
nypost.com	75	31
moat.com	54	34
cbc.ca	49	33
oracle.com	49	33
codebeautify.org	45	33
breitbart.com	41	33
foursquare.com	40	30
... 70 more	38 – 79	20 – 34

(A) Signature correlation: **addthis.com**

Domain	LCS	Cat.
dell.com	80	27
nike.com	80	27
vmware.com	80	27
adobe.com	77	27
foxnews.com	77	27
tiktok.com	77	27
ea.com	77	27
fedex.com	77	27
aeroflot.ru	77	27
easyjet.com	66	23
... 200 more	60 – 80	20 – 27

(B) Signature correlation: **nasdaq.com**

TABLE 5.7: When correlating fingerprint signatures of 10k sites, we found high LCS values for the addthis.com site, all connected to the ORACLE fingerprint service MOAT, and for nasdaq.com sites, which showed very high matches with a fingerprint network of similar size.

In further investigation of the fingerprint behavior, we looked at the fingerprint scripts of the websites themselves and found copyright protected comments on *addthis.com*. The company claiming copyright for the fingerprinting script is *Moat Inc.*, a digital advertising service whose parent company is *Oracle Corporation*. By chance, we correlated and found similar signatures on both company websites, which in this case implies that all collected fingerprint information will ultimately be under Oracle's control. The signature we find on *nasdaq.com* is, together with *coinbase.com*, the largest network we found from our smaller list. With over 200 entries of nearly identical or minor variations of the same fingerprint signature and a wide range of page topics, this network is one of the largest networks we found in our analysis. Very well-known sites with the same signature in this network range from the manufacturer *nike.com* the American media website *foxnews.com*, the delivery service *fedex.com*, the Russian airline *aeroflot.com*, the American video game company *ea.com*, the American tech company *dell.com* and *tiktok.com*, a popular Chinese video-sharing platform.

Since the behavior of the fingerprint on *thepiratebay.com* is unique with its aggressive and high range, we also searched for this signature. A more expected find, as shown in Figure 5.8, is the distribution of LCS signatures. We find more than 50 almost identical signatures with an LCS score of more than 70 on websites that are neither publicly known nor can be attributed to a company by us. Due to their content and the excessive use of advertising, we consider these sites to be rather dubious. A website offering illegal file-sharing services, such as *thepiratebay.com* does, is expected to have its LCS signatures correlate well with other websites offering dubious services.

Domain	LCS	Cat.
divxtotal.la	73	28
cb01.expert	73	28
mhometheater.com	73	28
www5.javmost.com	73	28
descargas2020.org	73	28
skidrow-games.com	73	28
thepiratebay10.org	73	28
thisav.com	73	28
tv.mylivecricket.biz	73	28
worldfree4u.ink	71	26
... 50 more	31 – 71	3 – 26

(A) Signature correlation: **thepiratebay.org**

Domain	LCS	Cat.
google.se	61	12
google.com.ly	61	12
google.co.ao	61	12
google.com.do	61	12
google.com.kw	61	12
google.com.uy	61	12
google.ca	31	7
google.com.hk	31	7
google.com.mx	31	7
google.com.eg	31	7
... 108 more	29 – 31	6 – 7

(B) Signature correlation: **google.com**

TABLE 5.8: When correlating the fingerprint signatures of 10k websites, we found high LCS values for thepiratebay.org, which are all dubious sites, and for google.com, sites that are only on the Google domain and therefore have an identical fingerprint.

On the other side of the table in Figure 5.8b we show the unexpectedly clean results we got from the domain *google.com*, where all 120 fingerprint signatures we found are all related to the domain Google. As we found out in Figure 5.5, this shows that Google uses its own fingerprinting techniques that are not shared with other websites in the way they use them. In addition, this result shows that even with comparatively lower similarities of 30 continuously measured categories and a category coverage value of 7, the accuracy and precision of our measurements are still high enough that not a single domain other than Google appeared in this range of recognized signatures.

5.4 Advanced Fingerprinting Techniques

In our analysis, we found different kinds of fingerprint behavior. On one hand, we located specifically targeted fingerprinter that perform actions on sub-directories, such as a website's shopping card, where we could measure differences in fingerprinting activity on nearly all websites (Figure 5.5). In addition, we identified fingerprinters that perform data extraction by chance and at random times on different sub-directories of a website, like seen on *breitbart.com*. Further, as shown in Figure 5.2, we found websites whose fingerprinter and their aggressiveness varied when clicking on a cookie banner.

Fingerprinting actions, as we have examined in our studies, are usually done covertly. The average user is not familiar with the technologies used for obtaining digital fingerprints and the possibilities where taking fingerprints might interfere with data privacy. For this reason, they are an easy target as they do not recognize that any harm has been done and no apparent threat exists. The obfuscation and complexity of the browser ecosystem offer many possibilities that even technically savvy people find it difficult to distinguish, whether a browser-related functionality could violate privacy. In recent years, many threads and discussions appeared in technology news and forums, such as HackerNews [87], where many concerns have been expressed about fingerprinting web devices and browsers like [88, 89, 90]. It is interesting to note that most of these well educated and more privacy-conscious people see the dangers of fingerprinting. However, what sparks their discussions on data protection is, that even professionals have no idea how to prevent such fingerprinting activities, nor have they developed ways to protect themselves from actions that lead to exposure of sensitive user data [91]. This underlines the fact that there is an apparent threat, especially for the average user who is less privacy-conscious, as companies easily violate privacy policies without being seen, hindered, or punished. We further found, that regardless of whether websites comply with regulations, such as the GDPR, and providing a transparent way to opt-out or accept their privacy policies, as seen in Figure 5.2, that many websites persist with aggressive fingerprinting activities when a user presses the *"Accept All"* button on a cookie banner.

Along investigating actual fingerprinters and ways how they are integrated into the functionality of a website, we created signatures to identify

websites that are potentially part of the same fingerprinting network. We monitored their activity and found out that any utilization and correlation of user fingerprints, in any case we measured, occurs on the server-side. For this reason, any process that goes beyond the generation and extraction of fingerprints is invisible to the public and to our techniques used by FPMON, which only monitors the client-side JavaScript code. Based on the extent of data extracted, as seen in B.1, it would be reasonable to assume, that a company behind each script that collects fingerprints, could correlate users and user data to a greater extent. Further, this could be done on a data set gathered from all websites of an entire fingerprint network that share the same fingerprint script or service.

One technique we have observed that could make the correlation of user and user fingerprints much easier for companies is the method used by *addthis.com*. The fingerprinter of addthis is integrated into a widget to share content with other websites, primarily social media. In this way, when sharing content, the fingerprint is taken together with an ID or reference to the user profile, which provides additional information such as the user's name or nickname. When sharing content, the widget most likely establishes a link between the uncorrelated fingerprint data and the profile of third-party social media, such as the profile created by *vk.com* or *twitter.com*. This way, the fingerprint data can be immediately identified and thus matched to a specific individual, since the fingerprint is linked to a specific account or profile ID. We observed that this behavior is one of the most intrusive and concerning techniques on the market regarding its extent, effectiveness, and stealth.

Another unexpected result we found is, that websites tend to perform fingerprinting on multiple positions inside a web-application, such as on login mechanisms or privacy-banner procedures, like we demonstrated in Study 2 (see Section 5.2). This leads to the conclusion, that there is probably even more fingerprint activity hidden within the functionalities of a website, as we have already examined with the widget of addthis.com. Furthermore, when investigating the internals of monitoring a website, we found identifiers that are temporarily stored in the local browser cache, consisting of additionally generated IDs for each specific fingerprint. Together with a timestamp, this additional metadata is added to the set of actual fingerprint data when sent to a third party. That way, we think that companies are most likely to index and validate a fingerprint before adding it to their fingerprint server-side correlation database. At *amazon.de*, for example, we have seen behavior after a page request, that loops with a certain delay and updates a fingerprint ID in the local browser memory, thus updating the generated fingerprint. We suspect that this could be related to a fingerprint behavior that continuously refreshes the stored data while waiting for the sending trigger to be activated in order to always have the most up to date fingerprint data prepared. It is also reasonable to assume that many of the larger websites have some form of fingerprint post-authentication in order to get the fingerprint script to resent every time a user performs a task that needs further authorization.

Companies that prevent fraud by linking behavioral and customer system data do important and responsible work, especially in the area of finance or

other businesses and services critical to the user. However, it is very difficult to tell whether a company's sole intention is to sell its fingerprint activities for fraud protection purposes or to use this information for other business purposes, such as a sale to third-party advertising networks, without the explicit consent of a user. A recent example of a website where we did not expect fingerprints, is the website of the NGO *savethechildren.org*. We have found evidence in the media, that websites like savethechildren.org are often targets of credit card fraud, where stolen credit card information is tested by donating a small amount of 2$ to the NGO [92]. Therefore, it could be a reasonable practice for NGOs to correlate the user and owner of a credit card, by linking browser sessions and configurations in order to prevent such actions in the future and facilitate the prosecution of credit card fraudsters.

5.5 Privacy

One of the key observations in all parts of our studies is, that the data flow of data generated by fingerprint services leads to and ends at a certain point, in most cases at large advertising agencies. We have shown in our research on fingerprinting behavior in Sections 5.2 and 5.3 that sites that perform fingerprinting actions are often not visibly connected to the companies that receive, process, and subsequently have knowledge about the extracted data. Due to non-transparent privacy practices, we conclude, that there is no real way for the user to verify the trustworthiness of the entire fingerprinting process, even if it is used in a non-intrusive and meaningful way that only enhances the user experience. Currently, transparency is enforced by law through cookie banners that display a website's current privacy practices and allow users to decide whether or not to accept the privacy policy. Under the GDPR regulation, it must be possible for any user to opt-out of the privacy policy and still have access to the service of this website [7]. Ignoring the banner technically did not affect the functionality of the website, but often results in half of the page being covered with repeated cookie banner requests. We have not fully investigated the methods cookie banners use to its full extend. However, we have spent enough time exploring that area, so we often came across privacy policies, such as those found on *addthis.com* and *ebay.com*, which have hundreds of affiliates and thousands of lines of policy text and cross-references to the policies of other companies, making it almost impossible to fully understand the whole process, especially for a user who simply wants to use the services of a website. We think, many websites deliberately make it difficult for the user to understand for what purpose and to whom their sensitive information is being sold or shared. We have seen several banners whose corresponding websites we will not explicitly disclose here. After visiting one of these websites, a user had to either accept the terms and policies or uncheck up to 10 pre-selected checkboxes to reject the terms before being granted access to the service. Obtrusive behavior like that is not only not transparent and dishonest to users, it is also far from being compliant with the GDPR regulation[6]. In addition, a user can not be sure whether those terms of privacy claims are true and whether those

companies stick to their promises. More importantly, our scans and manual measurements, as seen in Figure 5.2, have shown that even if a user rejects or ignores the privacy banner, the fingerprinting activity of websites often can be measured regardless. However, the above-mentioned behavior suggests that the area remains rather a grey area alongside the existing rules, where lack of transparency and exploitation of user confidence is not sanctioned.

Privacy-conscious users who want to increase the transparency and privacy of their browsers to limit the spread of their fingerprints and sensitive data often use extensions to improve data protection, as we examined in our study in Section 5.1. We have shown in Table 5.3, that using these extensions has a useful and positive impact on privacy where it prevents the aggressive use of advertisements and gathering of fingerprints. However, the caveat of using such extensions is, that they may increase user confidence too much, although only parts of the privacy risks are addressed. That way, users could develop a false sense of security and privacy while still performing sensitive operations, such as banking or shopping on dubious websites. We think that software that claims that its service keeps users fully secure, is not only an unethical marketing tool to generate higher revenues, but can also be seen as one of the real pitfalls, as it undermines transparency, security, and privacy consciousness of users.

5.6 Limitations

Through our investigation of the fingerprinting activities of various websites, we gained insight into the general behavior of a typical fingerprinter. Even though fingerprinting involves the extraction of a multitude of data, from the point of a website, not all information from the data set of a fingerprint gathered can be used to uniquely identify an individual. Some parts of the fingerprint, such as an image generated from a canvas element, have bigger variations in its uniqueness than information, such as the available number of CPU cores. We noticed that the individual parts of the fingerprint data are assembled, as seen in the implementation of fingerprintjs2 [40], and then either saved in the local browser memory or sent directly to the website or third-parties after activating a trigger mechanism. Our FPMON extension, however, focuses on the calculation of a fingerprint and does not cover how fingerprints are stored and incrementally updated on a local machine. In addition, the FPMON gives no indication of which trigger mechanisms are used for exfiltration.

The main limitation of the FPMON browser extension, however, is the pre-defined and, therefore, limited coverage of the fingerprint categories it recognizes. The coverage of the fingerprint categories mentioned is a metric that reflects our understanding of the behavior of fingerprints in different ways, on which fingerprints are generated and thus recognized. Although, an approach like that can only represent a subset of the fingerprint functionality to visualize what happens and is possible in reality. We believe that in the real world, there are much more complex and even stealthier ways to generate and extract fingerprint data from the JavaScript toolset and browser

ecosystem, especially from companies that do not make their source code publicly available. The purpose of FPMON and FPCRAWL is not to comprehensively identify all of them, but rather to show the presence of aggressive and not so aggressive fingerprinting techniques on every website and web application tested as accurately as possible.

The main limitation regarding our crawler FPCRAWL is that for complexity reasons we cannot cover the full JavaScript functionality when scanning multiple websites. To achieve completeness, one would have to use every possible function available in a web application to be able to make a valid statement. For this reason, we decided to scan just the landing pages of 10.000 websites, as described in the Sections 5.1 and 5.2, to get an idea about the fingerprint activity on a large scale.

In contrast to scanning landing pages with FPCRAWL, FPMON can still be used manually to perform an in-depth analysis of entire websites and web applications. Furthermore, the FPMON extension also gives us insight into the details of the implementation, and shows the actual function calls and return values of individual fingerprint tasks. Further, we note, while examining the unprocessed metrics of FPMON when developing the extension, that a fingerprint or parts of fingerprints can be remarkably similar to other fingerprints from many websites. However, it is not possible for us and, therefore, for the user, to determine to what extent exposed fingerprint information has a negative impact on user privacy. Nevertheless, it has been proven that there are categories, such as canvas or audio fingerprints, that may solely recognize a particular user by correlation with another fingerprint [93] without the need for any other fingerprint data.

In a scenario where a user changes hardware components, such as the graphics or sound card, updates the computer's operating system or browser, changes the IP address, or uses a VPN, the company processing the fingerprints must be able to detect any change in the fingerprint data and link it back to the appropriate user with the new fingerprint. In this case, the company first needs to find out if the extracted data is related to a known user. If not, they create an entry for a new user in the fingerprint database. Still, to the best of our knowledge, it should be highly unlikely for a website to know with a certainty of 100%, whether a new user can be linked unambiguously to a fingerprint in the fingerprint database. This is because websites can only calculate probabilities and match signatures on that basis even though single fingerprint parts have very high entropy. Manipulating the system with software and hardware updates causes a certain amount of fuzziness and noise in the fingerprint data, which repeatedly causes changes in the signature. Therefore, we believe, that the more different unique fingerprints can be associated with a single tracked user, the higher should be the probability of identifying a known user in a set of fingerprint data. However, we also believe that this drastically increases the false positive and false negative rate of correct user identification, since there is no single key fingerprint category which tells a website it must be part of a fingerprint and thus identifies a particular user.

Chapter 6

Conclusion and Future Work

In the last chapter, we will briefly summarize the results and give an overview of the tasks that we have managed to solve with this thesis. Furthermore, we will discuss topics and open problems that can be explored in future work.

6.1 Summary

During our analysis in previous studies, we encountered many expected and unexpected ways in which fingerprinting of websites is used to collect data on visiting users. The typical approach to perform fingerprinting on most of the websites we scanned is to dynamically load a JavaScript script, either from the company's own content delivery network or from a third-party CDN.

To study internal behavior, we built the FPMON browser extension and the FPCRAWL crawler to further expand our view of fingerprinting methods beyond the recent research by Vastel *et al.* [73] and Laperdrix *et al.* [76]. The FPMON extension overrides various fingerprint-related browser functions, as shown in Chapter 3, and monitors their execution and behavior at runtime. During the development of this extension, we found that FPMON is a very good tool for enhancing users' knowledge of what functionality is running in real-time on a website. Therefore, we optimized the extension for performance and transparency, so that it can be used without a noticeable impact on the page load time behavior of a website. FPMON allows users to see how the website is performing sensitive and aggressive fingerprinting actions that could violate the user's privacy. In contrast, the FPCRAWL takes advantage of the functionality of the extension and extends the scope in a fully automated and scalable way to scan huge lists of websites with a functional and active DOM of a browser.

As previously pointed out by Snyder *et al.* [72], the range of possible fingerprint categories that exist in the real world is unknown and continues to increase. However, in this thesis, we have identified, measured, and analyzed 40 different categories, almost twice as many categories as the open-source implementation of a full-fledged fingerprint covered by fingerprintjs2 (Table 5.1). As shown in Figure 5.5, Both the extension and crawler environments work precisely to even find differences and similarities in the implementation of each measured fingerprint category and cover a much wider range of fingerprint detection compared to current research, as conducted in

[46, 76, 73]. We compared the various privacy-enhancing browser extensions and settings, such as *Adblock*, *EFF Privacy Badger Tool*, *DuckDuckGo Privacy Essentials*, *Firefox Strict Mode* and visualized their impact on different websites (Tables 5.3, 5.1). We validated, as previously pointed out by Laperdrix *et al.* in [76], that the most commonly used method of these extensions to protect users from third-party trackers is to blacklist the respective tracker script. If a tracker is detected by one of these privacy-enhancing extensions, the execution and consequently the entire functionality of the tracker script is blocked. Apart from this, the FPMON Browser extension has measured little to no impact on the decrease in fingerprint activity in these privacy-enhancing extensions.

One of the main objectives was to perform a large-scale scan to examine the distribution of the fingerprint categories used. To achieve this, we performed a scan of 10.000 websites and a scan with a link depth of one, covering all subpages of a website on a smaller list of 45 sites. We found that approximately 90% of all the websites we scanned perform actions that could be part of the fingerprinting behavior, while 30% of these websites perform aggressive to very aggressive actions (Table 5.2), and then created a list of fingerprint categories that highlight their distribution, as shown in Figure 5.3. The top five categories most commonly used for the capture of fingerprints are *UserAgent*, *Screen and Window* related calls, *Content Language*, *List of Plugins* and *Storage* (Section 5.2). We created signatures of the fingerprint activities we measured and correlated them by calculating their respective LCS scores to make the similarities of the signatures visible (Section 5.3). With the LCS metric, we were able to reliably distinguish between the noise of frequently used DOM functionality and unique fingerprint signatures that were used repeatedly on multiple websites, as shown in the Tables 5.6, 5.7, and 5.8. We found similarities on many websites and discovered corporate fingerprint networks, including a media network that is surprisingly present on all popular German news sites, a network that can only be found on Google domains, the network behind *thepiratebay.com* and Oracle's advertising network *MOAT*.

6.2 Conclusion

As the Internet is free and we have no control over the coding and privacy practices of websites, ways must be found to deal with potentially abusive behavior. The method we chose with the FPMON extension and the FPCRAWL crawler is educational, as we believe that knowledge on the user side and transparency about the website, is the only effective way to deal with the risk of disclosing sensitive information. We built a way for the average user that acts like the browser's internal warning message, such as when visiting potential unsafe websites, that alert users by giving a short warning of a potential threat. In our opinion, self-reflection at the right moment helps to improve the awareness of a specific threat. We decided that the colors of a traffic light plus a short aggressiveness indicator as the extension's badge text (Figure 4.2.2) and a more detailed view for the experienced user in an opening extension popup, is an appropriate way to aggregate the information we get from the monitored fingerprint execution environment. This way, users can see what is happening on a website in real-time and increases transparency and security through self-control, without having to know the entire technological stack behind the fingerprint capturing process.

Although we may be able to turn off fingerprinting, with all its advantages and disadvantages, to a certain extent (Section 6.3), we encourage every party, users and websites, to understand that an informed and transparent user base is the greater lever to improve privacy than another tool in the fingerprint arms race research. A browser extension or configuration that improves privacy, as we have seen and analyzed in Study 1 (Section 5.1), has a measurable and positive impact on the user's privacy because it performs a task that, to some degree, prevents or disables the fingerprinting activity of a website. The ability to change the behavior of websites in a privacy-enhancing way is especially useful because it provides a level of flexibility, to adjust settings to our needs, that we have seen in the past with advertisement blocking extensions [94]. While we were able to measure a positive effect of privacy-enhancing extensions, we also noted that this could give a false sense of security and protection of personal data. Even if such an extension covers all privacy-violating techniques at a given point in time, it effectively will only work for the period until new techniques are found.

However, our approach, which points out that user information must always be treated with the necessary care with regard to data protection, offers transparency instead of implementing another way of circumventing potentially abusive functionality. Telling websites and ad-networks that users have insight into the fingerprint functionality and behavior of a website creates the desired pressure on a website not to defame themselves for getting caught doing potentially harmful actions. Since websites and users act in a customer-like relationship, in which trust plays an essential role, it should be crucial to run and maintain a healthy business, or things could likely happen like in "The Fall of Facebook" [95], where a chain of data breaches and other events uncovered broad violations of the applied privacy policies, and subsequently showed deliberate disregards of user rights.

6.3 Future Work

The main goal of this thesis is to provide a proof of concept to show and visualize current JavaScript and fingerprint related activities. For the use case of our monitoring mechanism with FPMON, we defined the capabilities to detect certain fingerprinting function calls and attributes. Since all browsers and, therefore, all executed JavaScript code runs on a user-owned and controlled machine, FPMON is also capable of controlling and manipulating any code that is executed when a defined callback event is triggered. Therefore, it would be possible to inject arbitrary data into fingerprint parameters, even before the fingerprinting script itself encrypts and subsequently exfiltrates the data. Manipulating the fingerprint and, with this, void the integrity of the data, effectively making it unusable, could be a valuable contribution to research in this field.

Another area where further research could lead to interesting results could be the analysis of third party scripts as a whole, rather than, as we have done, focusing on entire web applications with all scripts included. In order to further improve the capabilities of FPMON, additional and more detailed callback functions could be added to the extension. In this way, each category or attribute could be defined with a much finer granularity to further distinguish between exactly the same or similar implementations. Thus, it would also be possible to deactivate parts of fingerprint actions and manipulating and detecting them by behavior.

Perhaps the most promising area that we have not covered in this thesis is the in-depth analysis of web applications with FPMON. Many websites build complex and customized web applications, whose internal functionality, e.g. everything that happens after the login to a web app, remains hidden to us in this work, but can still be analyzed by using our extension. In addition, privacy browsers like the Tor Browser could be examined to see if they, while visiting websites, restrict JavaScript, and consequently fingerprinting to the extent they promise.

References

[1] James Slevin. "Internet". In: *The Blackwell encyclopedia of sociology* (2007).

[2] *Wikipedia, the free encyclopedia.* https://www.wikipedia.org. Accessed: 2020-06-09.

[3] *Reddit: the front page of the internet*. https://www.reddit.com. Accessed: 2020-06-09.

[4] *Twitter. It's what's happening.* https://www.twitter.com. Accessed: 2020-06-09.

[5] Jonathan R Mayer and John C Mitchell. "Third-party web tracking: Policy and technology". In: *2012 IEEE Symposium on Security and Privacy*. IEEE. 2012, pp. 413–427.

[6] *Browser-fingerprinting using the AudioContext and Canvas API.* https://www.eff.org/de/deeplinks/2018/06/gdpr-and-browser-fingerprinting-how-it-changes-game-sneakiest-web-trackers. Accessed: 2020-05-20.

[7] *Cookies, the GDPR, and the ePrivacy Directive.* https://gdpr.eu/cookies/. Accessed: 2020-05-23.

[8] *If browsers are the new operating systems, why don't they have the security to match?* https://www.zdnet.com/article/if-browsers-are-the-new-operating-systems-why-dont-they-have-the-security-to-match/. Accessed: 2020-06-09.

[9] *Osi model Reference*. https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/. Accessed: 2020-06-09.

[10] Christoph Alme. *Web browsers: An emerging platform under attack*. 2008.

[11] *2019 Magecart Timeline.* https://www.rapidspike.com/blog/2019-magecart-timeline/. Accessed: 2020-06-08.

[12] Sergey Brin, James Davis, and Hector Garcia-Molina. "Copy detection mechanisms for digital documents". In: *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*. 1995, pp. 398–409.

[13] Kenneth Todd Wease. *Target-based SMB and DCE/RPC processing for an intrusion detection system or intrusion prevention system*. 2012.

[14] Antoine Vastel et al. “FP-STALKER: Tracking browser fingerprint evolutions”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 728–741.

[15] Guoqiang Shu and David Lee. “Network protocol system fingerprinting a formal approach”. In: *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*. IEEE. 2006, pp. 1–12.

[16] Martin Husak et al. “HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting”. In: *EURASIP Journal on Information Security* 2016.1 (2016), p. 6.

[17] Eric Kollmann. “Chatter on the Wire: A look at DHCP traffic”. In: *Online]. Avaliable: http://myweb. cableone. net/xnih/download/chatter-dhcp. pdf* (2007).

[18] Fedor V Yarochkin et al. “Xprobe2++: Low volume remote network information gathering tool”. In: *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*. IEEE. 2009, pp. 205–210.

[19] Sumeet Singh et al. “Automated Worm Fingerprinting.” In: *OSDI*. Vol. 4. 2004, pp. 4–4.

[20] Robert Beverly. “A robust classifier for passive TCP/IP fingerprinting”. In: *International Workshop on Passive and Active Network Measurement*. Springer. 2004, pp. 158–167.

[21] Matthew Smart, G Robert Malan, and Farnam Jahanian. “Defeating TCP/IP Stack Fingerprinting.” In: *Usenix Security Symposium*. 2000.

[22] Lloyd G Greenwald and Tavaris J Thomas. “Toward Undetected Operating System Fingerprinting.” In: *WOOT* 7 (2007), pp. 1–10.

[23] Robert Beverly and Arthur Berger. “Server siblings: Identifying shared IPv4/IPv6 infrastructure via active fingerprinting”. In: *International Conference on Passive and Active Network Measurement*. Springer. 2015, 149 bibrangedash 161.

[24] Omar E Elejla et al. “IPv6 OS Fingerprinting Methods”. In: *International Visual Informatics Conference*. Springer. 2017, pp. 661–668.

[25] Quirin Scheitle et al. “Large-scale classification of IPv6-IPv4 siblings with variable clock skew”. In: *2017 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE. 2017, pp. 1–9.

[26] Ofir Arkin. “A remote active OS fingerprinting tool using ICMP”. In: *login: the Magazine of USENIX and Sage* 27.2 (2002), pp. 14–19.

[27] Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. “Remote physical device fingerprinting”. In: *IEEE Transactions on Dependable and Secure Computing* 2.2 (2005), pp. 93–108.

[28] Daisuke Takahashi et al. “IEEE 802.11 user fingerprinting and its applications for intrusion detection”. In: *Computers & Mathematics with Applications* 60.2 (2010), pp. 307–318.

[29] Wolfgang John and Sven Tafvelin. "Analysis of internet backbone traffic and header anomalies observed". In: *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. 2007, pp. 111–116.

[30] *Nmap Network Scanning*. https://nmap.org/book/man-os-detection.html. Accessed: 2020-05-03.

[31] *P0f the highly scalable and extremely fast identification of the operating system and software on both endpoints of a vanilla TCP connection.* https://lcamtuf.coredump.cx/p0f3/. Accessed: 2020-05-05.

[32] Gabi Nakibly, Gilad Shelef, and Shiran Yudilevich. "Hardware fingerprinting using HTML5". In: *arXiv preprint arXiv:1503.01408* (2015).

[33] *The Baseline Interpreter: a faster JS interpreter in Firefox 70.* https://hacks.mozilla.org/2019/08/the-baseline-interpreter-a-faster-js-interpreter-in-firefox-70/. Accessed: 2020-03-26.

[34] *The Document Object Model (DOM).* https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. Accessed: 2020-03-24.

[35] *JavaScript Obfuscator Tool: A free and efficient obfuscator for JavaScript.* https://obfuscator.io/. Accessed: 2020-04-02.

[36] *HTTP State Management Mechanism.* https://tools.ietf.org/html/rfc6265. Accessed: 2020-02-12.

[37] *Using HTTP cookies.* https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. Accessed: 2020-02-12.

[38] *Web Fundamentals, Web Storage API, Storage Taxonomy.* https://developers.google.com/web/fundamentals/instant-and-offline/web-storage. Accessed: 2020-02-15.

[39] *Are you unique? Investigate the diversity of fingerprints.* https://amiunique.org/fp. Accessed: 2020-05-10.

[40] *Fraud detection API.* https://fingerprintjs.com/. Accessed: 2020-05-11.

[41] *Modern and flexible browser fingerprinting library.* https://github.com/Valve/fingerprintjs2. Accessed: 2020-05-11.

[42] Ralph C Merkle. "A fast software one-way hash function". In: *Journal of Cryptology* 3.1 (1990), pp. 43–58.

[43] *Browser Fingerprinting: An Introduction and the Challenges Ahead.* https://blog.torproject.org/browser-fingerprinting-introduction-and-challenges-ahead. Accessed: 2020-04-06.

[44] Shun-Wen Hsiao et al. “Combining dynamic passive analysis and active fingerprinting for effective bot malware detection in virtualized environments”. In: *International Conference on Network and System Security*. Springer. 2013, pp. 699–706.

[45] Ariel Rabkin. “Personal knowledge questions for fallback authentication: Security questions in the era of Facebook”. In: *Proceedings of the 4th symposium on Usable privacy and security*. 2008, pp. 13–23.

[46] Pierre Laperdrix et al. “Morellian Analysis for Browsers: Making Web Authentication Stronger with Canvas Fingerprinting”. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2019, pp. 43–66.

[47] Nikolaos Karapanos et al. “Sound-proof: usable two-factor authentication based on ambient sound”. In: *24th USENIX Security Symposium (USENIX Security 15)*. 2015, pp. 483–498.

[48] Joseph K Liu et al. “Fine-grained two-factor access control for web-based cloud computing services”. In: *IEEE Transactions on Information Forensics and Security* 11.3 (2015), pp. 484–497.

[49] David A Redberg. *Transfering soft tokens from one mobile device to another*. US Patent App. 15/851,250. 2019.

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

[51] Vacha Dave, Saikat Guha, and Yin Zhang. “Measuring and fingerprinting click-spam in ad networks”. In: *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. 2012, pp. 175–186.

[52] Nick Nikiforakis et al. “Cookieless monster: Exploring the ecosystem of web-based device fingerprinting”. In: *2013 IEEE Symposium on Security and Privacy*. IEEE. 2013, pp. 541–555.

[53] *What percentage of browsers with javascript disabled?* `https://blockmetry.com/blog/javascript-disabled`. Accessed: 2020-05-09.

[54] *Web HTML Element noscript.* `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/noscript`. Accessed: 2020-05-09.

[55] *Using the Web with JavaScript turned off.* `https://www.smashingmagazine.com/2018/05/using-the-web-with-javascript-turned-off/`. Accessed: 2020-05-09.

[56] Craig E Wills and Doruk C Uzunoglu. “What ad blockers are (and are not) doing”. In: *2016 Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*. IEEE. 2016, pp. 72–77.

[57] Christopher Soghoian. “Why private browsing modes do not deliver real privacy”. In: *Center for Applied Cyber security Research, Bloomington* (2011).

[58] *Pi-hole Network-wide Ad Blocking.*
`https://pi-hole.net/`. Accessed: 2020-05-14.

[59] *Panopticlick 3.0: Is your browser safe against tracking?*
`https://panopticlick.eff.org/`. Accessed: 2020-05-25.

[60] Peter Eckersley. "How unique is your web browser?" In: *International Symposium on Privacy Enhancing Technologies Symposium*. Springer. 2010, 1–18.

[61] *Electronic Frontier Foundation: Defend free speech. Fight surveillance. Support innovation.*
`https://eff.org/`. Accessed: 2020-05-25.

[62] Jonathan R Mayer. "Any person... a pamphleteer": Internet Anonymity in the Age of Web 2.0". In: *Undergraduate Senior Thesis, Princeton University* (2009), p. 85.

[63] Gunes Acar et al. "FPDetective: dusting the web for fingerprinters". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, pp. 1129–1140.

[64] Antoine Vastel et al. "FP-scanner: the privacy implications of browser fingerprint inconsistencies". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 135–150.

[65] *Protections Against Fingerprinting and Cryptocurrency Mining Available in Firefox Nightly and Beta.*
`https://blog.mozilla.org/futurereleases/2019/04/09/protections-against-fingerprinting-and-cryptocurrency-mining-available-in-firefox-nightly-and-beta/`. Accessed: 2020-05-19.

[66] Shujiang Wu et al. "Rendered Private: Making GLSL Execution Uniform to Prevent WebGL-based Browser Fingerprinting". In: *28th USENIX Security Symposium (USENIX Security 19)*. 2019, pp. 1645–1660.

[67] Erik Trickel et al. "Everyone is different: client-side diversification for defending against extension fingerprinting". In: *28th USENIX Security Symposium (USENIX Security 19)*. 2019, pp. 1679–1696.

[68] Pierre Laperdrix, Benoit Baudry, and Vikas Mishra. "FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques". In: *International Symposium on Engineering Secure Software and Systems*. Springer. 2017, pp. 97–114.

[69] Naoki Takei et al. "Web browser fingerprinting using only cascading style sheets". In: *10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*. IEEE. 2015, pp. 57–63.

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

[71] Oleksii Starov and Nick Nikiforakis. "Xhound: Quantifying the fingerprintability of browser extensions". In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, pp. 941–956.

[72] Peter Snyder, Cynthia Taylor, and Chris Kanich. "Most websites don't need to vibrate: A cost-benefit approach to improving browser security". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 179–194.

[73] Antoine Vastel et al. "FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers". In: *NDSS Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb'20)*. 2020.

[74] Babak Amin Azad et al. "Short Paper-Taming the Shape Shifter: Detecting Anti-fingerprinting Browsers". In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2020, pp. 160–170.

[75] Kevin Bock et al. "unCaptcha: a low-resource defeat of recaptcha's audio challenge". In: *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*. 2017.

[76] Pierre Laperdrix et al. "Browser fingerprinting: a survey". In: *ACM Transactions on the Web (TWEB)* 14.2 (2020), pp. 1–33.

[77] *EFF Privacy Badger Extension.* `https://privacybadger.org/`. Accessed: 2020-05-22.

[78] *Browser-fingerprinting using the AudioContext and Canvas API.* `https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32016R0679`. Accessed: 2020-05-20.

[79] *Market Share Statistics for Internet Technologies.* `https://www.netmarketshare.com/`. Accessed: 2020-05-25.

[80] *Selenium automates browsers.* `https://pypi.org/project/selenium/`. Accessed: 2020-03-02.

[81] *Overview of Docker Compose.* `https://docs.docker.com/compose/`. Accessed: 2020-03-02.

[82] *V8: Google's open source high-performance JavaScript and WebAssembly engine.* `https://v8.dev/`. Accessed: 2020-03-03.

[83] *Adblock blocks ads and pop-ups on YouTube, Facebook, Twitch, and your favorite websites.* https://getadblock.com/.

[84] *DuckDuckGo: Privacy Protection For Any Device.* `https://chrome.google.com/webstore/detail/duckduckgo-privacy-essent/bkdgflcldnnnapblkhphbgpggdiikppg?hl=en`. Accessed: 2020-05-22.

[85] *Content blocking with firefox strict mode.* `https://support.mozilla.org/en-US/kb/content-blocking`. Accessed: 2020-05-22.

[86] *The top sites on the web.* `https://www.alexa.com/topsites`. Accessed: 2020-05-23.

[87] *YCombinator; HackerNews*. https://news.ycombinator.com/news. Accessed: 2020-06-02.

[88] *HackerNews Discussion: Browser Fingerprinting: A Survey.* https://news.ycombinator.com/item?id=19838123. Accessed: 2020-06-02.

[89] *HackerNews Discussion: Block Fingerprinting with Firefox.* https://news.ycombinator.com/item?id=20054831. Accessed: 2020-06-02.

[90] *HackerNews Discussion: Audio Fingerprinting using the AudioContext API.* https://news.ycombinator.com/item?id=21436414. Accessed: 2020-06-02.

[91] *HackerNews Discussion: Am I Unique?* https://news.ycombinator.com/item?id=22148512. Accessed: 2020-06-02.

[92] *Didn't make that donation to charity? Watch out for fraud.* https://www.creditcards.com/credit-card-news/charity-donation-fraud-1267/. Accessed: 2020-06-02.

[93] *Browser-fingerprinting using the AudioContext and Canvas API.* https://audiofingerprint.openwpm.com/. Accessed: 2020-05-28.

[94] Matthew Malloy et al. "Ad blockers: Global prevalence and impact". In: *Proceedings of the 2016 Internet Measurement Conference*. 2016, pp. 119–125.

[95] *The Fall of Facebook.* https://medium.com/s/story/the-fall-of-facebook-part-i-6ac2d6a744c5. Accessed: 2020-06-02.

Appendix A

Source Code

```
const observer = new MutationObserver(mutations => {
  mutations.forEach(({
    addedNodes
  }) => {
    addedNodes.forEach(node => {
      // Find <script> tags
      if (node.nodeType === 1 && node.tagName === 'SCRIPT') {
        // Whitelisting our own code to be not be affected
        if (!(node.hasAttribute("FMON_WHITELIST"))) {
          // Disable async/defer on all scripts and third-parties
          deactivateAsync();
          deactivateDefer();
        }
      }
    })
  })
})

// Starts the monitoring of <script> tags
observer.observe(document.documentElement, {
  childList: true,
  subtree: true
})
```

FIGURE A.1: DOM observer to surveil HTML script tags.

The FPMON browser extension and FPCRAWL crawler source code are released with the submission of this thesis and can be retrieved from the following repositories on GitHub.

- **FPMON Extension**: https://github.com/KybranzF/fpmon
- **FPCRAWL Crawler**: https://github.com/KybranzF/fpcrawl

Appendix B

Fingerprint

Category	Fingerprint Data
userAgent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.87 Safari/53
webdriver	not available
language	en-US
colorDepth	24
deviceMemory	8
hardwareConcurrency	16
screenResolution	1920,1080
availableScreenResolution	1920,1080
timezoneOffset	-60
timezone	Europe/Berlin
sessionStorage	true
localStorage	true
indexedDb	true
addBehavior	false
openDatabase	true
cpuClass	not available
platform	Linux x86_64
plugins	Chromium PDF Plugin,Portable Document Format, application/x-google-chrome-pdf,pdf,[...]
canvas	canvas winding:yes,canvas fp:[...]
webglVendorAndRenderer	NVIDIA Corporation~GeForce GTX 970/PCIe/SSE2
adBlock	false
hasLiedLanguages	false
hasLiedResolution	false
hasLiedOs	false
hasLiedBrowser	false
touchSupport	0,false,false
fonts	Arial,Bitstream Vera Sans Mono,Courier,Courier New, Helvetica,Times,Times New Roman
audio	124.04344890356151

TABLE B.1: Complete user fingerprint generated by fingerprintjs2.