BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

PREVENTION OF PASSIVE BROWSER FINGERPRINTING

PREVENCE PŘEDÁVÁNÍ PASIVNÍHO OTISKU PROHLÍŽEČE

TERM PROJECT
SEMESTRÁLNÍ PROJEKT

AUTHOR **Bc. JOZEF HRUŠKA**
AUTOR PRÁCE

SUPERVISOR **Ing. LIBOR POLČÁK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2023

Abstract

This diploma thesis aims to design a new protection layer against passive browser fingerprinting, which complements existing protection against active fingerprinting in the JShelter web extension. The protection layer changes the values of selected HTTP headers and modifies the web browser environment to avoid inconsistencies.

Abstrakt

Cílem této diplomové práce je navrhnout novou vrstvu ochrany proti pasivnímu otisku prohlížeče, která komplementuje již existující ochranu proti aktivnímu otistku ve webovém rozšíření JShelter. Navržená ochranná vrstva upravuje hodnoty vybraných HTTP hlaviček a zároveň modifikuje protředí webového prohlížeče aby zamezila případným nekonzistencím.

Keywords

fingerprint, browser fingerprint, passive browser fingerprint, privacy protection, web extension, JShelter, JavaScript, HTTP

Klíčová slova

digitální otisk, otisk prohlížeče, pasivní otisk prohlížeče, obrana soukromí, webové rozšíření, JShelter, JavaScript, HTTP

Reference

HRUŠKA, Jozef. *Prevention of Passive Browser Fingerprinting.* Brno, 2023. Term project. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Libor Polčák, Ph.D.

Prevention of Passive Browser Fingerprinting

Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

. . . .
Jozef Hruška
March 5, 2023

Acknowledgements

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant apod.).

Contents

Chapter 1

Introduction

Although more than 30 years have passed since the invention of the web back in 1989 [15], it is still an integral part of our lives, possibly more than ever before. People worldwide use the web daily to work, learn, consume content and connect with others. Every day our smartphones, tablets, laptops, or even smartwatches and fridges connect to the web.

This diverse ecosystem of devices created a need to identify users who use them uniquely. The ability to uniquely identify a user on a single device or across multiple devices allows applications to personalize the content they provide or improve their security. One of the methods of user identification is browser fingerprinting, the collection of data from the browser to create a fingerprint of the device. Just as a person's fingerprint can uniquely identify that specific person, given that we have enough data, we can uniquely identify a device and its user and act on this information.

However, gathering this kind of data can be considered a violation of privacy [14], which is the primary motivation behind creating privacy-first browsers and browser extensions to fight against fingerprinting. One of the browser extensions set on improving privacy is JShelter[1] which focuses on preventing active fingerprinting.

Active fingerprinting is a method of obtaining a fingerprint, typically by running JavaScript code to collect data from browser APIs [21]. The other variant is passive fingerprinting, in which the data is collected solely from network traffic, for example, HTTP headers.

This work aims to design and implement a browser extension to prevent the collection of passive fingerprints by altering data in HTTP requests and explore possibilities of integrating this protection into JShelter. It is important to note that the altered data must be consistent with the data obtainable through active fingerprinting methods. Otherwise, the fingerprinting application could recognize an attempt to mitigate the effects of fingerprinting.

Fingerprinting is an ever-evolving field. Chapter 3 describes its nature, history, and different approaches to collecting a fingerprint. The last two sections of chapter 3 provide an outlook on threats and opportunities fingerprinting brings to the table because, as we know, nothing is purely black and white.

As this work explores the capabilities of mitigating fingerprinting with the help of browser extensions, the following chapter 2.4 gives the reader an insight into the inner working of extensions.

---

[1] JShelter is an anti-malware browser extension to mitigate potential threats from JavaScript, including fingerprinting [4] (Available at https://jshelter.org/).

Finally, the last chapter 5 proposes a design of passive fingerprinting protection that works consistently with active solutions to keep user-observable side effects to a minimum.

1.1 Definitions

- **Fingerprint**: A collection of device-specific characteristics gathered for the purpose of identification.
- **Fingerprinting**: The process of collecting the device information (data) to compute a fingerprint. Terms fingerprinting, device fingerprinting, and browser fingerprinting will be used interchangeably in the following chapters.
- **Fingerprinter**: An application performing fingerprinting, typically on a third-party remote server.

Chapter 2

Web browsers

In this chapter, the first section, 2.4.1, describes the WebExtensions API exposed by a browser to give extensions more capabilities. Recently, significant changes to the ecosystem of extensions were announced with the introduction of Manifest V3, described in section 2.4.2. The last section, 4, is dedicated to the JShelter extension on which this work builds.

2.1 History of web browsers

Web browsers have come a long way since the early days of the World Wide Web. From their humble beginnings as basic applications for viewing text-only web pages, modern browsers have evolved into the powerful tools we know today. Following paragraphs outline the evolution of browsers, highlighting a few of the most significant features added over the years.

The first browser, named WorldWideWeb[1], was created at CERN by Tim Berners-Lee only a year after he published the first World Wide Web proposal [15]. The very early browser aimed to demonstrate how the user-facing layer of the WWW (browsers) could work in the future. The browser was only available on the NeXTSTEP[2] operating system and had limited features.

One of the earliest features added to browsers was support for images. The Mosaic browser[3], released in 1993, was the first to allow users to view images on the web, followed by support for tables, which allowed for more complex web page layouts.

As the web gained popularity, developers needed a way to make web pages more interactive and dynamic. The birth of JavaScript programming language addressed this need. JavaScript was first introduced in 1995[4] by Netscape Communications Corporation to add interactivity to static HTML pages. Since then, it has become one of the world's most widely used programming languages, with a vast ecosystem of libraries and frameworks[5].

Nowadays, web browsers expose a variety of JavaScript APIs that allow web developers to interact with the browser and its underlying components, such as the operating system or even the hardware. While it is true that these APIs gave the developers much-needed possibilities, they come at the cost of compromised privacy, described in detail in the following chapters.

[1] https://worldwideweb.cern.ch
[2] https://en.wikipedia.org/wiki/NeXTSTEP
[3] https://en.wikipedia.org/wiki/Mosaic_(web_browser)
[4] https://en.wikipedia.org/wiki/Netscape_Navigator_2
[5] https://www.npmjs.com/

2.2 JavaScript

In its original form, JavaScript is a client-side language executed in the user's web browser rather than on a web server [18]. It allows applications to respond to user actions in real time without making a round trip to the server. However, the rise of popularity and the fact that JavaScript is a relatively developer-friendly language meant that, eventually, runtimes that allowed running JavaScript outside the browser emerged. The most popular JavaScript runtimes include Node.js[6], Deno[7], or Bun[8].

2.2.1 ECMAScript

As the web was rapidly gaining popularity, developers started using JavaScript to create dynamic and interactive web pages. However, there was no standardization for the language, meaning different browsers implemented JavaScript differently. This made it difficult for developers to create web applications that worked consistently across different platforms and browsers [18].

In 1996, Netscape Communications Corporation, the company behind the Netscape Navigator web browser, submitted JavaScript to ECMA International[9]. This standards organization develops and maintains standards for several technologies, including programming languages. The goal was to create a standardized language version that could be implemented across different platforms and browsers.

ECMA International established a committee to develop a standardized scripting language, which was initially called ECMA-262. The standard's first version, ECMAScript 1, was released in June 1997. Since then, ECMAScript has undergone several revisions, the latest being ECMAScript 2022. Each standard version adds new features and capabilities to the language while maintaining backward compatibility with previous versions [18].

Although ECMAScript is technically the correct name for the language, JavaScript is used in the following chapters as the industry uses this name predominantly.

2.2.2 Web APIs

Web applications can use JavaScript Web APIs to interact with various web browser features and functionality. The number of available APIs depends on the browser. For example, at the time of the writing, Firefox had more than 100 APIs[10], with more APIs still being added. Here is a short selection of Web APIs important to the topic of this thesis [22, 28]:

- **Canvas API**: Provides a way to dynamically create and manipulate graphics and animations in a web page using JavaScript.
- **Navigator**: Provides information about the web browser or user agent in which the code runs.
- **Web Audio API**: Provides advanced audio processing and synthesis capabilities, allowing developers to create interactive audio experiences on the web.

[6] https://nodejs.org/
[7] https://deno.land/
[8] https://bun.sh/
[9] https://www.ecma-international.org/
[10] https://developer.mozilla.org/en-US/docs/Web/API

- **WebGL**: Allows for creating and rendering interactive 3D and 2D graphics within a web browser using the computer's graphics hardware capabilities.

All the APIs listed above are commonly used in fingerprinting to identify devices and their users uniquely [21]. Fingerprinting, the main focus of this thesis, is described more in-depth in the following chapter.

2.3 HTTP

HTTP (Hypertext Transfer Protocol) is a request-response protocol that allows web browsers to request resources from web servers and to receive responses that include the requested resources [25].

When a user types a URL into a web browser's address bar, the browser sends an HTTP request to the server hosting the resource. This request typically contains information such as the request's method, the URL of the requested resource, and any HTTP headers or other metadata associated with the request. The server then resolves the request and sends a response containing the response status code and body, among other things.

Web applications can also ask the browser to send HTTP requests on their behalf with the help of XMLHttpRequest (XHR)[11] or Fetch API[12] APIs. Web applications commonly transfer only minimal HTML and JavaScript with the first request to improve performance. Once loaded, this skeleton can load additional resources if needed.

2.3.1 Headers

The client and the server use HTTP headers[13] to transfer additional information with the HTTP request or response. Headers consist of a case-insensitive name and a value separated by a colon (":"). HTTP defines standardized headers for content negotiation, authentication, or passing additional context [27]. The protocol also allows the use of entirely custom headers.

When an application sends a request, the browser automatically includes default headers. Web applications can add additional headers or overwrite the default ones.

2.4 Browser extensions

Browser extensions, or add-ons, can modify and enhance the capability of a browser [8]. Users install extensions for various reasons, such as to increase security or productivity, remove ads, or modify the browser UI to their liking. Each browser implements extensions slightly differently, but to a large extent, most browsers follow the WebExtensions API standard.

This section describes the WebExtensions API standard mentioned before and the recently introduced Manifest V3, which significantly affects the ecosystem of browser extensions.

[11] https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest
[12] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
[13] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

2.4.1 WebExtensions API

WebExtensions API, first introduced by Google in chromium-based browsers, is a set of special-purpose APIs the browser provides to installed extensions. Firefox[14], Safari[15], and Opera[16] have become widely compatible with the WebExtensions API, making it the standard across all major web browsers. This broad adoption allowed developers to create cross-browser extensions with minor changes to the original codebase.

WebExtensions API provides a wide range of functionality for building browser extensions, including access to browser tabs and windows, manipulating web pages and user settings, and interacting with other extensions and websites [3].

WebExtension API differs from the standard JavaScript Web APIs described in the previous section. Applications use the standard Web APIs to interact with the content and functionality of a web page. In contrast, extensions use the WebExtensions APIs to interact with the browser itself. Some of the interfaces provided by the WebExtensions API are [3]:

- **Cookies API**: Allows to query and modify cookies. Extensions can also configure a callback function browser calls whenever a particular cookie changes.
- **DeclarativeNetRequest API**: Allows to block or modify network requests by specifying declarative rules. This API allows extensions to modify requests without giving them direct access to their content.
- **History API**: Allows interaction with the browser's record of previously visited pages. Extensions can add, remove, and query for URLs in the browser's history.
- **Tabs API**: Allows interaction with the browser's tab system. Extensions can create new, modify, or rearrange existing tabs.

The WebExtensions API is easy to use and provides consistent interfaces and functionality across different browsers. Lately, a push for increased security and privacy rules for extensions resulted in Manifest V3, described more in-depth in the following section.

2.4.2 Manifest V3

Manifest V3 represents one of the most significant shifts in the extensions platform since its launch [2]. According to the team behind the Chrome browser, it is an answer to raising concerns about privacy, security, and performance of browser extensions. The most notable changes compared to its predecessor Manifest V2 are [2]:

- **Service workers**: Manifest V3 replaces background pages[17] with extension service workers, similar to traditional web service workers.
- **Network request modification**: In the outgoing Manifest V2, extensions could intercept and modify requests procedurally. In contrast, in the Manifest V3, extensions have to define rules and ask the browser engine to modify requests on their behalf. The primary motivation behind this change was privacy, as extensions could previously access requests with little or no limitations.

[14] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions

[15] https://developer.apple.com/documentation/safariservices/safari_web_extensions

[16] https://dev.opera.com/extensions/apis

[17] Manifest V2 extensions used background pages to run a single persistent script to manage tasks, or a state of the extension [1].

- **Support for Promises**: The Chrome team added Promise support to some WebExtensions APIs so that extension developers can use modern JavaScript features such as promise chains or async/await patterns.

Although welcomed from a privacy-concerned standpoint, these changes made it more difficult for developers to create extensions to protect from fingerprinting, mainly because of the network request modification differences.

```json
[
  {
    "id": 10,
    "priority": 2,
    "action": {
      "type": "modifyHeaders",
      "responseHeaders": [
        {
          "header": "h1",
          "operation": "remove"
        },
        {
          "header": "h2",
          "operation": "set",
          "value": "v2"
        },
        {
          "header": "h3",
          "operation": "append",
          "value": "v3"
        }
      ]
    },
    "condition": {
      "urlFilter": "headers.com/123",
      "resourceTypes": ["main_frame"]
    }
  }
]
```

Listing 2.1: An example of a declarative rule which modifies selected response headers [2].

Listing 2.1 shows an example of the new declarative rule syntax. The rule defined above modifies request headers, specifically removing a header named `h1`, changing the value of `h2`, and adding a new header, `h3`.

Chapter 3

Browser fingerprinting

Fingerprinting is a process of obtaining information about a user's device to construct a digital marker called a fingerprint. Applications can construct a fingerprint from various data, such as information about the device hardware, operating system, browser, or even seemingly unimportant information, such as installed fonts or a language preference.

Browser fingerprinting [21] is a subcategory where the application tries to create a fingerprint from the information available in a browser. As the main focuses of this theses are browsers and browser extensions, the term fingerprinting will be used predominantly over browser fingerprinting for ease of reading.

Information to form a fingerprint may be collected either actively, typically by running a JavaScript code inside a browser, or passively by monitoring the network communication.

The first section, 3.1, briefly tells the history of browser fingerprinting, from the time it was explored for the first time to its current form. Section 3.2 describes the most common fingerprinting countermeasures, and the third section, 3.3, compares the active and passive fingerprinting approaches fingerprinting applications use to collect data. The last section, 3.4, summarizes the threats and opportunities fingerprinting brings to the table.

3.1 History of browser fingerprinting

Laperdrix et al. provide excellent insight into the history of browser fingerprinting [21]. It all started in 2009 when Jonathan Mayer first investigated if differences presented by browsers of various users could be used to identify these users uniquely. He conducted an experiment where he collected the contents of the navigator, screen, `Navigator.plugins`, and `Navigator.mimeTypes` objects of browsers that connected to the website of his experiment. Of 1328 clients, he uniquely identified 1278 (96.23%).

A year later, in 2010, Peter Eckersley pushed this research further when he conducted a Panopticlick experiment [17]. In this experiment, he obtained over 470 thousand fingerprints, uniquely identifying 83.6% (94.2% with Flash or Java enabled).

Although only a short time has passed since these two pioneering works were published, fingerprinting became a standard in the industry thanks to easy-to-use solutions like FingerprintJS[1], which has an open-source core. However, this broad adoption of fingerprinting solutions brought severe security and privacy implications [14].

[1] https://fingerprint.com

3.2 Fingerprinting countermeasures

Although not straightforward, to a certain degree, it is possible to fight against fingerprinting. Fingerprinting countermeasures are typically categorized followingly [19, 23]:

- **Creating a homogenous fingerprint**: Fingerprinting tries to find differences between devices to identify them. Hypothetically, if every device in the world had the same characteristics (and thus the same fingerprint), identification would not be possible because each device would be indistinguishable from others. However, to create absolute homogeneity, every valuable device characteristic exposed to the fingerprinting application must be the same. If one or more characteristics differ, the anonymity set breaks into smaller ones, making the identification possible again.
- **Altering the fingerprint for each domain to prevent cross-domain tracking**: Values of characteristics typically used to create a fingerprint are adjusted for each domain, making it challenging to track the device across multiple domains. Algorithms performing these adjustments are doing so in a seemingly random way to avoid patterns the fingerprinting applications could use for identification.
- **Altering the fingerprint for each session to prevent cross-session tracking**: Similar to the previous cross-domain countermeasure, but values of characteristics are changed for each browsing session. In other words, the fingerprint is different whenever the browser is closed and reopened again.
- **Detecting and blocking fingerprinting**: An attempt to detect fingerprinting in real-time, block access to the data, or limit the page's ability to upload the fingerprint. This approach is strict and intrusive as it can limit the functionality of the visited application, or, in the worst scenario, it could break the page completely.

Applications implementing these countermeasures typically combine two or more categories for better overall protection [19]. The type of application is also important. Browsers are more capable of combating fingerprinting as they are closer to the operating system than browser extensions.

3.3 Approaches to fingerprinting

Fingerprinting is typically divided into two categories – active and passive fingerprinting. This section briefly explains both, including examples of data that can be collected and the danger of inconsistencies that the fingerprinting application can detect.

3.3.1 Active fingerprinting

When a fingerprinter has to perform explicit actions to obtain the information it needs, it is called active fingerprinting. By actions, we typically mean running a code to collect the data from a remote machine (i.e., port scanning) or directly on the targeted device (i.e., running a JavaScript code in a browser).

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML,
    like Gecko) Version/15.6 Safari/605.1.15
```

Listing 3.1: An example of Safari User-Agent string.

Running a JavaScript code in a browser is a potent tool to collect data about the software and hardware on the device. Modern browsers provide sets of APIs to give web applications native-like capabilities, often giving developers access to detailed data. Among the most noteworthy interfaces used for fingerprinting is the `Navigator` interface [9], which provides information about the application running the script, usually the browser or the `Screen` interface [10], which represents the screen on which the current window is rendered.

Name	Source (JavaScript API)	Example value
Available memory	`Navigator.deviceMemory`	`8`
Battery level	`BatteryManager.level`	`0.94`
Cookies enabled	`Navigator.cookieEnabled`	`true`
Number of log. processors	`Navigator.hardwareConcurrency`	`8`
Preferred languages	`Navigator.languages`	`["en-US", "en-GB"]`
PDF display support	`Navigator.pdfViewerEnabled`	`true`
Screen width	`Screen.width`	`1920`
Screen height	`Screen.height`	`1080`
User-Agent string	`Navigator.userAgent`	See listing 3.1.

Table 3.1: An example of data available from JavaScript APIs of a browser.

Table 3.1 shows examples of information obtainable through browser APIs. The fingerprinting applications prefer characteristics that change less frequently or are entirely static. For example, the battery percentage will likely change often, which can affect the resulting fingerprint. On the other hand, the number of logical processors will presumably remain the same for the device.

3.3.2 Passive fingerprinting

Intuitively, passive fingerprinting is the alternative method to active fingerprinting. In passive fingerprinting, the information required to create a fingerprint is collected from characteristics observable in the network traffic, which means there is no need to execute a data-gathering code directly in the browser. Because no code is executed on the fingerprinted device, the fingerprinting is not observable from it, making it difficult to detect an attempt at fingerprinting.

Name	Source (HTTP/HTTPS header)	Example value
Available memory	`Device-Memory`	`8`
Content encoding	`Content-Encoding`	`gzip, deflate, br`
Preferred languages	`Accept-Language`	`en-US,en-GB,en;q=0.9`
Referer address	`Referer`	`https://amiunique.org/`
User-Agent string	`Navigator.userAgent`	See listing 3.1.

Table 3.2: An example of data available in HTTP request headers.

Table 3.2 shows examples of data directly available in request headers. These HTTP headers and other characteristics are used to create a passive device fingerprint.

3.3.3 Inconsistencies

Fingerprinting countermeasures described in section 3.2 often modify browser characteristics to affect the fingerprint. Suppose a characteristic can be obtained by active as well as passive fingerprinting, meaning it has multiple data sources. In that case, the countermeasure algorithm must apply these modifications to all data sources to avoid introducing inconsistent pieces of information. A piece of inconsistent information has a valid but conflicting value between various data sources [30].

Among the examples shown in tables 3.1 and 3.2 are characteristics with multiple data sources, namely the available memory, preferred languages, and the User-Agent string. Values of specific characteristics, such as the available memory or the User-Agent string, are one-to-one copies meaning the countermeasure algorithm can apply the same transformation to all data sources. However, this is not the case for the values of preferred languages characteristic. Preferred languages transferred in HTTP headers include the quality factor[2] for each language tag, but it is absent from the value obtainable from the JavaScript API. Tables 3.1 and 3.2 illustrate this difference. An algorithm cannot apply the same transformation for all data sources as it has to be aware of these differences in representation.

Suppose the countermeasure algorithm misapplies the transformation to a data source, or the data source is left unchanged entirely. In that case, the fingerprinting application can detect and act on this attempt at fingerprint manipulation.

3.4 Threats and opportunities of fingerprinting

Fingerprinting is predominantly known for its often privacy-invading practices, but as with most things in our world, nothing is purely black and white. Fingerprinting brings good things (opportunities) and bad things (threats) to the table. This section discusses both, including example use cases.

3.4.1 Threats

Although fingerprinting is also used for good, it became popular mainly for reasons which could be considered a violation of privacy [14]. A person could be tracked for a long time until the device's characteristics change enough to affect the fingerprint. During this period, applications can show ads or products personalized to users based on browsing history and activity to boost their sales.

3.4.2 Opportunities

The ability to uniquely identify a device and transitively the user of the device allows third parties to change the behavior of their services and applications. Applications can use fingerprinting for payment fraud prevention and detection [13] to prevent fraudsters from using stolen credit, debit, or checking data.

[2]Quality weights or quality values are used in HTTP headers to specify the order of priority in a comma-separated list (`https://developer.mozilla.org/en-US/docs/Glossary/Quality_values`).

Chapter 4

JShelter

In 2019, Martin Timko designed and implemented a proof of concept browser extension called "Javascript Restricter" as a part of his master's thesis [29]. Inspired by his work, Libor Polčák et al. [19] created JShelter, a browser extension focused on fingerprinting prevention, limitations of rich web APIs, prevention of attacks connected to timing, and learning information about the device, the browser, the user, and surrounding physical environment and location. JShelter offers versions for three major browsers - Firefox, Chrome, and Opera.

4.0.1 JavaScript Shield

The pivotal protection feature of JShelter is the JavaScript Shield [5], which modifies the behavior of the JavaScript environment of the browser. JShelter provides fake information to confuse fingerprinters and make webpage-triggered attacks harder or impossible.

The JavaScript Shield internally consists of small independent pieces of code called wrappers, which modify the original behavior of JavaScript APIs available in the browser. The behavior of wrappers can be categorized followingly [5]:

- **Precision reduction**: The original value is unnecessarily precise for most use cases. JavaScript Shield modifies the values so that typical and essential use cases are unaffected.
- **Providing fake information**: Some wrappers provide fake information mostly to confuse fingerprinters. For example, canvas wrappers modify the image so that the exact instructions produce a different result in each session and for each domain.
- **Hiding information**: JavaScript Shield hides information from selected browser APIs that are generally unnecessary or have little use. Depending on the API, it might return an error, an empty value, or block the API completely.

To modify the values of the browser characteristics, the JavaScript Shield uses a technique called *farbling* [19]. This technique is based on the implementation of farbling in Brave [16], which has its roots in previous privacy research, including the PriVaricator [23] and FPRandom [20].

Farbling first deterministically generates a per-session, per-eTLD+1[1] seed. This seed is later used to slightly randomize the output of web APIs typically used for fingerprint-

[1] eTLD+1 is a part of a domain name, including TLD, eTLD and one more domain part (More on https://jfhr.me/what-is-an-etld-+-1/).

ing. The way a seed is generated makes cross-site and cross-session tracking difficult for fingerprinting applications.

Chapter 5

Design proposal

This work aims to design and implement a new protective shield for the JShelter extension that can misrepresent information commonly used for passive fingerprinting. The first section of this chapter emphasizes the importance of avoiding alterations that could disrupt the user experience. Each of the following sections describes a selected HTTP header, proposes a modification to its value, the advantages and disadvantages of this modification, and necessary changes to the browser APIs to maintain consistency.

5.1 Minimizing the impact on users

It is crucial to remember that every modification the new protective shield will make to the outgoing requests can impact the response received from the server. For example, changing the preferred language reported by the browser to English would help make the fingerprint look more homogenous, as English is the prevalent language on the Internet. However, if the server reacts and returns a website in English, the user may need help understanding its content.

5.2 HTTP header: User-Agent

The User-Agent request header is a characteristic string that lets servers and network peers identify the application, operating system, vendor, and/or version of the requesting user agent [11]. The value of this header varies depending on the operating system and browser type and version, making it an invaluable component of a fingerprint.

```
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/51.0.2704.103 Safari/537.36
```

Listing 5.1: An example of Chrome User-Agent string [11].

```
Mozilla/5.0 (iPhone; CPU iPhone OS 13_5_1 like Mac OS X) AppleWebKit/605.1.15
    (KHTML, like Gecko) Version/13.1.1 Mobile/15E148 Safari/604.1
```

Listing 5.2: An example of Safari User-Agent string (mobile version) [11].

Figures 5.1 and 5.2 show examples of User-Agent strings of Chrome and Safari, which according to Statcounter GlobalStats [12], are the top two browsers with the highest market share of 64.62% and 18.29%, respectively. Statcounter GlobalStats is collecting statistics from more than 1.5 million sites with yearly views ranging from 5 to 6 billion. The examples above show that some browsers and operating systems have version numbers following the semantic versioning specification [26] or its variation. Semantic versioning in the most recent version 2 is composed of three segments separated by a dot (“.”), where:

- The first segment (**MAJOR**) is incremented when an incompatibility, such as a breaking API change, is introduced.
- The second segment (**MINOR**) is incremented when new functionality is added while still maintaining backward compatibility.
- The last segment (**PATCH**) is incremented for all other changes, for example, a bug fix or code refactoring.

From this, it is possible to make an observation. Given two version numbers that differ only in the last segment (**PATCH**), we know these software builds should have the same functionalities, meaning they should be indistinguishable for the user.

With this hypothesis, a change (either incrementing or decrementing) of the **PATCH** segment value would likely affect the fingerprint. However, at the same time, it is unlikely that the user would notice any difference.

The algorithm cannot change version numbers randomly, as this could create versions that were or never would be released. Instead, it would keep a list of previously released versions gathered from the changelog[1] of the specific software product. The algorithm would then randomly select a suitable version from the list according to the requirements described above.

5.2.1 Consistency

The User-Agent string is also available through the `Navigator` interface, with the value being identical to the value transferred in the `User-Agent` HTTP header. It is necessary to apply the same transformation to the `Navigator.userAgent` to avoid inconsistencies.

5.3 HTTP header: Accept

The Accept request HTTP header indicates which content types the client can understand, expressed as MIME types [6]. It is possible to add or remove MIME types to affect the fingerprint. However, the server may react to this modification and respond with an unexpected and incompatible response body.

```
Accept: text/html, application/xhtml+xml, application/xml;q=0.9, image/webp,
    */*;q=0.8
```

Listing 5.3: An example of Accept header contents [6].

[1] Software products typically maintain a list of published versions. For example, the changelog of Chrome is available at https://chromereleases.googleblog.com/search/label/Stable%20updates.

A viable solution is to remove certain MIME types, which would not result in incompatibility, but only a slight performance hit. For example, it is possible to remove modern image formats such as `image/webp` and force the browser to use older image formats. This modification would affect the header value (and, transitively, the fingerprint value) without potentially breaking the website.

5.3.1 Consistency

Browsers no longer offer a standard API for quickly checking supported MIME types. The `Navigator.mimeTypes` API that partially allows this check has been deprecated and removed from the relevant web standards [9]. A fingerprinting application could maintain a list of browser versions and supported MIME types which it would use to compare with the MIME types in the `Accept` HTTP

Maintaining consistency but also checking the consistency is not trivial when it comes to the supported MIME types. Therefore, no browser API modifications are necessary.

5.4 HTTP header: Accept-Language

The Accept-Language request HTTP header [7] indicates the natural language and locale that the client prefers. It is a list of preferred language tags separated by commas (","). A language tag consists of a 2-3 letter base language tag that indicates a language, optionally followed by additional subtags separated by a dash ("-"). The country or region variant (`en-US` or `fr-CA`) is the most common extra information stored in subtags. Language tags can optionally have a quality value (weight) represented by a decimal number separated from the tag by a semicolon (";").

```
Accept-Language: fr-CH, fr;q=0.9, en;q=0.8, de;q=0.7, *;q=0.5
```

Listing 5.4: An example of Accept-Language header contents [7].

Modifying this header is not straightforward because the server might respond with content in a different language if done wrong. One option is to change or remove language regions, as this information is often not crucial as the difference between language forms is usually manageable. For example, changing `en-US` (United States English) to `en-GB` (Great Britain English) might get noticed by users, but presumably, they would still understand the content.

5.4.1 Consistency

The client-preferred languages are also available through the `Navigator` interface under the `Navigator.languages` property. The languages are ordered the same way as in the `Accept-Language` header but without the quality value (weight). The first language is also extracted to the `Navigator.language`. Any change made to the `Accept-Language` HTTP header has to be reflected in these two properties of the `Navigator` interface.

5.5 List of HTTP headers

Another valuable characteristic used to create a fingerprint is a list of all headers present in a request. Most requests share standard headers such as those mentioned in previous sections 5.2, 5.3, and 5.4. However, it is common for requests to include less standard, even custom[2] headers.

```
Accept, Accept-Encoding, Accept-Language, Connection, Cookie, Host, User-Agent
```

Listing 5.5: An example of a list of headers present in an HTTP request.

It is possible to add new headers to the HTTP request to alter the resulting fingerprint. If the newly added headers are custom, this change would likely not affect the response.

[2] Until 2012, developers tended to prefix custom headers with “X-” to differentiate them from standardized headers, but this naming convention has been officially discouraged in RFC6648 [24].

Bibliography

[1] *Chrome Developers - Manifest V2* [online]. Google LLC [cit. 2022-12-30]. Available at: https://developer.chrome.com/docs/extensions/mv2/.

[2] *Chrome Developers - Manifest V3* [online]. Google LLC [cit. 2022-12-30]. Available at: https://developer.chrome.com/docs/extensions/mv3/intro/.

[3] *Chrome Developers - WebExtensions API reference* [online]. Google LLC [cit. 2022-12-30]. Available at: https://developer.chrome.com/docs/extensions/reference/.

[4] *JShelter* [online]. JShelter [cit. 2022-12-30]. Available at: https://jshelter.org/.

[5] *JShelter - JavaScript Shield* [online]. JShelter [cit. 2022-12-30]. Available at: https://jshelter.org/levels/.

[6] *Mozilla Developer Network - Accept HTTP header* [online]. Mozilla Corporation [cit. 2022-12-30]. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept.

[7] *Mozilla Developer Network - Accept-Language HTTP header* [online]. Mozilla Corporation [cit. 2023-01-1]. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Language.

[8] *Mozilla Developer Network - Browser extensions* [online]. Mozilla Corporation [cit. 2022-12-30]. Available at: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions.

[9] *Mozilla Developer Network - Navigator Web API* [online]. Mozilla Corporation [cit. 2023-01-9]. Available at: https://developer.mozilla.org/en-US/docs/Web/API/Navigator.

[10] *Mozilla Developer Network - Screen Web API* [online]. Mozilla Corporation [cit. 2023-01-9]. Available at: https://developer.mozilla.org/en-US/docs/Web/API/Screen.

[11] *Mozilla Developer Network - User-Agent HTTP header* [online]. Mozilla Corporation [cit. 2022-12-30]. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent.

[12] *Startcounter GlobalStats* [online]. [cit. 2023-01-6]. Available at: https://gs.statcounter.com.

[13] *Use Cases* [online]. FingerprintJS, Inc. [cit. 2022-12-30]. Available at: https://fingerprint.com/use-cases/.

[14] Article 29 Working Party. Opinion 9/2014 on the application of Directive 2002/58/EC to device fingerprinting. [online]. European Commission. [cit. 2022-12-30]. Available at: https://ec.europa.eu/justice/article-29/documentation/opinion-recommendation/files/2014/wp224_en.pdf.

[15] Berners Lee, T. *Information Management: A Proposal* [online]. 1989 [cit. 2022-12-30]. Available at: https://www.w3.org/History/1989/proposal.html.

[16] Brave Privacy Team. *Brave browsers - Fingerprinting defenses 2.0* [online]. Brave Software, Inc. [cit. 2023-01-10]. Available at: https://brave.com/privacy-updates/4-fingerprinting-defenses-2.0/.

[17] Eckersley, P. How Unique is Your Web Browser? In:. Berlin, Heidelberg: Springer-Verlag, 2010 [cit. 2022-12-30]. PETS'10. ISBN 3642145264.

[18] Flanagan, D. *JavaScript: the Definitive Guide : Activate Your Web Pages*. O'Reilly Media, Incorporated, 2011 [cit. 2023-02-19]. Available at: https://ebookcentral.proquest.com/lib/vutbrno/detail.action?docID=686420.

[19] L. Polčák, G. M. R. H. and McMahon, M. *JShelter: Give Me My Browser Back*. arXiv, 2022 [cit. 2022-12-30]. DOI: 10.48550/ARXIV.2204.01392. Available at: https://arxiv.org/abs/2204.01392.

[20] Laperdrix, P., Baudry, B. and Mishra, V. FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In: *ESSoS 2017 - 9th International Symposium on Engineering Secure Software and Systems*. Bonn, Germany: [b.n.], July 2017 [cit. 2023-01-6]. Available at: https://hal.inria.fr/hal-01527580.

[21] Laperdrix, P., Bielova, N., Baudry, B. and Avoine, G. Browser Fingerprinting: A survey. arXiv. 2019, [cit. 2022-12-30]. DOI: 10.48550/ARXIV.1905.01051. Available at: https://arxiv.org/abs/1905.01051.

[22] Mueller, J. P. *Security for Web Developers : Using JavaScript, HTML, and CSS*. O'Reilly Media, Incorporated, 2015 [cit. 2023-03-05]. Available at: https://ebookcentral.proquest.com/lib/vutbrno/detail.action?docID=4333783.

[23] Nikiforakis, N., Joosen, W. and Livshits, B. PriVaricator: Deceiving Fingerprinters with Little White Lies. In: *Proceedings of the 24th International Conference on World Wide Web*. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2015 [cit. 2023-01-6]. WWW '15. DOI: 10.1145/2736277.2741090. ISBN 9781450334693. Available at: https://doi.org/10.1145/2736277.2741090.

[24] P. Saint Andre, D. C. and Nottingham, M. *Deprecating the "X-" Prefix and Similar Constructs in Application Protocols* [Internet Requests for Comments]. RFC. RFC Editor, june 2012 [cit. 2023-01-6]. Available at: https://www.rfc-editor.org/rfc/rfc6648.

[25] Pasquali, S. *Mastering Node.js*. Packt Publishing, Limited, 2011 [cit. 2023-02-19]. Available at: https://ebookcentral.proquest.com/lib/vutbrno/detail.action?docID=1389335.

[26] Preston Werner, T. *Semantic Versioning Specification* [online]. [cit. 2022-12-30]. Available at: https://semver.org.

[27] R. Fielding, M. N. E. and J. Reschke, E. *HTTP Semantics* [Internet Requests for Comments]. RFC. RFC Editor, june 2022 [cit. 2023-01-6]. Available at: https://www.rfc-editor.org/rfc/rfc7231.

[28] Schauer, M. *Oblíbenost JavaScriptových API internetového prohlížeče*. Brno, CZ, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: https://www.fit.vut.cz/study/thesis/23312/.

[29] Timko, M. *Vylepšení rozšíření pro omezení volání JavaScriptu*. Brno, CZ, 2019. [cit. 2022-12-30]. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: https://www.fit.vut.cz/study/thesis/21824/.

[30] Vondráček, T. *Získávání informací o uživatelích na webových stránkách*. Brno, CZ, 2021. [cit. 2022-01-08]. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: https://www.fit.vut.cz/study/thesis/23972/.