BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

COMPARISON OF PRIVACY PRESERVING TOOLS IN WEB BROWSERS AND EXTENSIONS

POROVNÁNÍ NÁSTROJŮ ZAMEZUJÍCÍCH SLEDOVÁNÍ UŽIVATELE PROHLÍŽEČE

TERM PROJECT
SEMESTRÁLNÍ PROJEKT

AUTHOR **Bc. VOJTĚCH FIALA**
AUTOR PRÁCE

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

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

163497

Institut:	Department of Information Systems (DIFS)
Student:	**Fiala Vojtěch, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Computer Networks
Title:	**Comparison of Privacy Preserving Tools in Web Browsers and Extensions**
Category:	Web
Academic year:	2024/25

Assignment:

1. Study web browsers, privacy enhancing browser extensions. and techniques that prevent user tracking.
2. Learn options of controlled web browsers and extension testing.
3. Design an environment for repeatable and deterministic testing of the methods found to prevent user tracking.
4. Implement the proposed environment.
5. Test selected tools that you studied in point 1.
6. Evaluate achieved results and suggest possible future directions of the project.

Literature:

- PETRÁŇOVÁ, Jana. Porovnání webových rozšíření zaměřených na bezpečnost a soukromí. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií.
- PETRÁŇOVÁ, Jana. Běhová prostředí pro testování činnosti rozšíření pro webový prohlížeč. Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií.
- SMATANA, Andrej. Evaluation of Little Lies in Browser Fingerprinting. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií.

Requirements for the semestral defence:
Points 1-3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Polčák Libor, Ing., Ph.D.**
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	1.11.2024
Submission deadline:	21.5.2025
Approval date:	22.10.2024

Abstract

This thesis focuses on the comparison of tools that prevent the tracking of the browser user. It describes selected methods used to identify users and methods and tools that make this identification difficult. The thesis also describes selected existing methods and tools that can be used to compare different tools that prevent user tracking. Based on the existing methods, custom methods are also proposed for deterministic and repeatable testing of the tools. Proposed testing methods are implemented in the thesis to compare the performance of different tools based on defined criteria.

Abstrakt

Tato práce se zabývá srovnáním nástroju zamezujících sledovaní uživatele prohlížeče. Popisuje vybrané metody sloužící k identifikaci uživatelů a metody a nástroje, které tuto identifikaci znesnadňují. Práce dále popisuje vybrané existující metody a nástroje, které lze pro srovnání různých nástrojů zamezujících sledování uživatele využít. Na základě existujících metod jsou dále navrženy metody vlastní, které slouží k deterministickému a opakovatelnému testování nástrojů. Navržené metody testování jsou v rámci práce implementovány a umožňují srovnávat výkon různých nástrojů na základě definovaných kritérií.

Keywords

comparison, privacy, internet, browser extensions, browser, user tracking

Klíčová slova

porovnání, soukromí, internet, rozšíření prohlížeče, prohlížeč, sledování uživatele

Reference

FIALA, Vojtěch. *Comparison of Privacy Preserving Tools in Web Browsers and Extensions.* Brno, 2025. Term project. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Libor Polčák, Ph.D.

Comparison of Privacy Preserving Tools in Web Browsers and Extensions

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Ing. Libor Polčák, PhD. The supplementary information was provided by Ing. Ondřej Malačka. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Vojtěch Fiala
January 10, 2025

Acknowledgements

Here it is possible to express thanks to the supervisor and to the people which provided professional help

Contents

List of Figures

Chapter 1

Introduction

Website owners are motivated to track user activity across the Internet and understand who visits their page. One example is how advertisers utilize tracking to identify who views their advertisements [7]. Knowing who the person is enables them to provide more targeted advertisements, making it more probable that the person viewing the advertisement will purchase the shown item.

User tracking is present on many popular pages. Over two thousand [1] out of the ten thousand most popular websites have been observed to utilize multiple trackers. Since the trackers are used quite often, people may have valid concerns about their privacy. Many tools can be used to mitigate user tracking. These tools may be in the form of browser extensions or special browsers with built-in anti-tracking methods. Not only may these methods prevent user tracking, some of them may also improve the user experience. Some of the tools block HTTP requests to specified pages, thus reducing the total number of HTTP requests which can save both time and transferred bytes [21].

However, existing anti-tracking tools are mostly independent of each other and thus work differently. Therefore, users are presented with a choice of which anti-tracking tool to use. This thesis attempts to help the users with their choice by comparing multiple privacy-oriented tools in several different ways. Privacy-preserving tools are tested in a repeatable and deterministic way with limitations of each comparison clearly stated.

Chapter 2 explains what user tracking is and presents several mechanisms used to track the users. Chapter 3 presents existing tools and ways of comparing different privacy-oriented tools. Based on the existing approaches, new comparison methods are proposed in Chapter 4.

[1] https://www.ghostery.com/whotracksme

Chapter 2

User Tracking Methods and Mitigation

When a person visits a website, their activity is often tracked. That may be for many of reasons. These reasons can range from enhancing the user experience by remembering user's preferences (e.g., selecting white or dark display mode) to displaying advertisements tailored to user's interests. The pieces of code utilized for user tracking are referred to as *trackers* [4]. While the internal mechanisms of each tracker may vary, their objective remains the same – to identify the user who accessed the page.

This chapter introduces the reader to several tracking methods. Each method is presented along with possible ways to mitigate them. Afterward, the chapter explains general methods of tracking prevention. Finally, selected privacy-oriented web browsers and extensions are presented.

2.1 Tracking Techniques

A wide range of techniques exist to track users across the Internet. These techniques typically operate without the user's knowledge. The techniques can be classified into two basic categories, namely *stateful* and *stateless*, according to how they operate [16].

Stateful tracking techniques are methods that store information, such as user identification, on the user's computer for the specific purpose of tracking. Upon subsequent visits to the page, the identification is sent to the server, which allows the page, for example, to recall that the user is already logged in.

Stateless techniques attempt to uniquely identify the user without explicitly storing any data on the user's computer. The identification is based solely on the user's computer and browser properties, which the user unintentionally provides to the remote server.

This section presents the reader with specific techniques used to identify users. Furthermore, it explains how to counter these techniques and how the usage of countermeasures might affect the functionality of web pages.

2.1.1 Cookies

Cookies [3] are small data units stored on the user's computer to provide the stateless Hypertext Transfer Protocol (*HTTP*) with a method to retain information across multiple page visits. The data stored in cookies can only be accessed by the originating site. Cookies may be classified as either persistent or session-based. Persistent cookies remain stored on

the user's device after the browser is closed and expire after a pre-determined length of time. In contrast, session-based cookies are only stored until the browser or website is closed.

Using cookies can facilitate a more pleasant user experience by, for instance, storing user preferences, thereby eliminating the need for manual adjustments to the page on each visit. Another potential application is maintaining user authentication status, which can prevent the need for re-entering credentials on each page visit. The use cases mentioned above represent the intended functionality of cookies. However, it is also possible for cookies to be employed for the purpose of tracking users by storing a unique identifier which serves no other purpose. This identifier is then provided by the user to the page each time they revisit it.

First and Third-party Cookies

Cookies can be classified into two categories – those originating from the *first-party* and those originating from the *third-party* [9, 20]. When a user accesses a particular domain via a browser and the website on that domain employs cookies, the cookies set by that particular domain are referred to as first-party cookies. However, a page may also contain data originating from third parties. Suppose a page incorporates elements with third-party content originating from another page with a different domain name, which sets its own cookies. In that case, these cookies are designated as third-party cookies. Many elements may be used to load third-party content. Two example elements that utilize third-party content are the *img*[1] element, which can show remotely-hosted images, and the *iframe*[2] element, which allows a page to contain another page [17, 20].

Cookie Blocking

Many web browsers block all third-party cookies [20]. The reason is that third-party cookies are primarily employed to monitor user activity across multiple websites to offer more precision in targeted advertising.

Nevertheless, trackers can overcome this issue by utilizing first-party cookies [20]. Replacing cross-site third-party cookie tracking with first-party cookie tracking can be accomplished through many methods. For example, a page may share its first-party cookie with a third party by appending the identifier as a part of an HTTP request sent from the original page to a third-party tracker. However, linking arbitrary identifiers to a single user would be difficult. To address this problem, identifiers such as the user's email address are used. Users provide their email address to a web page to log in, and later, when they log in using the same email on another site, both sites may share the email address with a third-party tracker, which allows the tracker to recognize that the user has visited both sites.

One way to prevent first-party cookie tracking is to block all cookies entirely. However, this approach may severely limit the functionality of many websites. Alternatively, it would be possible to permit the usage of cookies but to delete them each time the browser is closed. However, this approach might not work for users who rarely close their browsers. These problems can be addressed by using sophisticated techniques to circumvent first-party cookie tracking. One such approach, based on machine learning, has been proposed by Munir et al. [20].

[1] https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img
[2] https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe

2.1.2 Evercookies

Evercookies [5] share the name with regular cookies mentioned above. Nevertheless, they work quite differently. Upon the removal of a conventional cookie, it is irretrievable, and the originating site has to set the cookie again. However, the removal of evercookies is a significantly more challenging process. These special cookies employ many techniques to evade deletion, such as using unconventional storage mechanisms or even multiple of them. Should the tracking data in one of the storage mechanisms be deleted, as long as at least one instance remains intact, evercookies may be able to use it to restore themselves to all the storage mechanisms from which they have been deleted.

Evercookies employ browser mechanisms to store themselves that were not designed for such purposes. Consequently, it becomes more challenging for users to remove the tracking data, as they may not be aware that it is being stored through such mechanisms. These mechanisms may include, for example, classic HTTP cookies, browser local storage, DOM properties, or browser cache. Historically, these storage methods included Flash cookies or Silverlight [5]. In order to mitigate this tracking technique, it is necessary to remove data from all possible storage or limit what storage can be accessed in the first place.

2.1.3 Browser Fingerprinting

The process of browser fingerprinting entails collecting data from the user's browser or the HTTP communication itself [16]. Contemporary fingerprinting techniques rely heavily on browser APIs to extract distinctive information, which is then employed to identify the user with high accuracy. The final fingerprint is derived from the collective data obtained through this process.

In order to prevent fingerprinting, three distinct approaches may be employed [23]:

- Creating homogenous fingerprints – All generated fingerprints are identical, and no individual can be considered distinctive, thus thwarting any potential for tracking.
- Modifying fingerprints on each visit – All fingerprints are modified (e.g., by randomization) to be different on every visited domain, thus making it impossible for trackers to link the fingerprints to a single user.
- Blocking fingerprinting attempts – Properties commonly utilized for fingerprinting are monitored, and should they be utilized by a tracker, network communication with such tracker is blocked. Another method is pre-emptively blocking all communication with known trackers.

The extraction of a fingerprint can be categorized according to the techniques employed, which can be broadly divided into two categories [23]: *passive fingerprinting*, which relies on the extraction of information from the HTTP header or the communication between a user and a server, and *active fingerprinting*, which extracts information from various browser APIs by using JavaScript or other technologies.

Passive Fingerprinting

The term *passive fingerprinting* is derived from the fact that the fingerprinting process does not require the utilization of any additional means of data collection. Upon accessing a website via a web browser, the user initiates an HTTP request, which is then sent to the

server. However, an HTTP packet also contains other data, such as the user's IP address. The HTTP and other protocol data are transmitted each time the user connects to a site, regardless of the user's intention. Without this data, the user would be unable to access the site.

Passive fingerprint may, for example, be comprised of the following elements: [16, 32]

- IP Address – used to identify the device in the TCP/IP Network layer.
- User-Agent[3] – HTTP header used to identify client's browser, operating system, and other information for compatibility purposes.
- Accept[4] – HTTP header to indicate what content the client can understand.
- Accept-Language[5] – HTTP header to indicate the language preference of the client.

In practical applications, trackers would employ a greater quantity of such data. Given that all of this data originates from the client, it is relatively straightforward to render them unusable, should we disregard possible consequences, by simply modifying them to another value. Such consequences may arise, for example, from modifying the IP address. If a client were to alter their IP address when sending a packet, simply changing it without implementing any additional modifications would break the routing protocols used to determine the destination for each packet, leading to communication failure.

In order to alter the IP address, tools such as Virtual Private Networks (*VPNs*) can be utilized. Rather than communicating directly with the website (or any other entity), all data is redirected to a VPN provider. This provider then effectively acts as an intermediary, directing each packet to its intended destination, thereby preventing third parties from discerning the original IP address, which would instead be replaced by the VPN provider's IP address. [11]

It can be reasonably assumed that correctly modifying an IP address will not result in any significant changes to a site's functionality. However, modifying HTTP headers may have a more pronounced impact on the behavior of a site. For instance, modifying the Accept-Language header, should the site support it, may result in the site being displayed in a different language.

Active Fingerprinting

In contrast to passive fingerprinting, active fingerprinting works by deploying the fingerprinting code directly to the user's browser. Actively acquiring browser data involves misusing various technologies provided by web browsers to create a fingerprint. One such way would be using the JavaScript *navigator*[6] interface. Other methods might be more insidious. This section will present several technologies that can be employed in the fingerprinting process.

Canvas Fingerprinting

A canvas element[7] is a programmable area that can be populated with graphics or text by the programmer. However, the same graphics drawn onto the element may appear

[3] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent
[4] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept
[5] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Language
[6] https://developer.mozilla.org/en-US/docs/Web/API/Navigator
[7] https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas

differently, depending on the user's system configuration, such as the installed graphical processing unit (GPU). This slight difference is the basis of canvas fingerprinting [19]. The canvas element also provides programmers with multiple API methods that can be employed in the process of fingerprint creation. These include `getImageData()`[8], which returns information about each pixel of the image, and `toDataURL()`[9], which returns an encoded representation of the canvas content. Extracting these data provides a strong and consistent fingerprint independent of other fingerprinting techniques.

The canvas fingerprinting methods work with WebFonts[10], which provide web pages with a simple method to include fonts [19]. Even when using the same browser and operating system on multiple machines, the results of rendering text on the canvas can vary due to minor differences in rendering engines. These subtle variations are what contribute to the generation of unique fingerprints.

Further research [1] revealed that this method could be extended even further by, for example, incorporating an additional graphic element or overlaying another text on the original.

Regarding countermeasures, the most straightforward approach with the most significant drawbacks would be to disable the element, or maybe even JavaScript itself, entirely. However, the missing support of the canvas element may itself prove to be an identifiable event. If the user wishes to retain the canvas functionality, they would need to utilize more sophisticated techniques that would modify the results returned by the methods associated with the canvas element. These techniques, for instance, involve making minor random alterations to the RGB values of chosen pixels, which, in the ideal scenario, should not even be perceptible to the user [26].

WebGL Fingerprinting

WebGL[11] technology allows users to render three-dimensional images. Fingerprinting based on this technology is similar to the Canvas fingerprinting described above, as it also employs the canvas element, thus enabling the same techniques to be employed for fingerprint calculation [19].

Further research [6] has demonstrated that a significant proportion of the WebGL rendering features can be employed to enhance the quality of the fingerprint. Additionally, WebGL itself provides a method[12] for retrieving data about the user's GPU, which may potentially constitute an additional component of the fingerprint. The techniques for preventing WebGL fingerprinting and their impact on the user experience remain the same as those employed for canvas fingerprinting. Additionally, some WebGL methods could be prevented from functioning to, for example, make it impossible for the page to acquire additional information about user's system, such as the mentioned GPU.

Font Fingerprinting

Fifield and Egelman [13] have demonstrated that individual letters or symbols utilizing the same font may be rendered differently depending on the browser used. Even when using the same browser and operating system, discrepancies can be observed depending on the system

[8] https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/getImageData
[9] https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/toDataURL
[10] https://developer.mozilla.org/en-US/docs/Learn/CSS/Styling_text/Web_fonts
[11] https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API
[12] https://developer.mozilla.org/en-US/docs/Web/API/WEBGL_debug_renderer_info

configuration. This method involves increasing the size of fonts drawn in the background and taking measurements of individual symbols. While it is less accurate than previous fingerprinting methods, it can still be used as a part of the final fingerprint. Another way to fingerprint users based on fonts is by obtaining the list of fonts installed on the user's system. Eckersley [10] has shown that the list of installed fonts can be extracted using JavaScript and presents a highly identifiable metric.

Mitigating fingerprinting attempts using this method may include randomizing the reported properties of shown symbols [13]. Another method would be disabling JavaScript, which is used for measurements or obtaining the list of fonts, and complete disabling of Cascading Style Sheets (*CSS*). Other than that, it is possible to force the web browser to work only with a predefined set of fonts shipped with the browser, thus limiting variability.

The mitigation methods may impact the user experience. For example, disabling both JavaScript and CSS would result in a significant change in the visual appearance of the majority of websites while also limiting their functionality. Similarly, using a predefined set of fonts may adversely affect the visual appearance of certain pages that were designed with a specific font in mind.

Cascading Style Sheets Fingerprinting

CSS can be utilized to recognize the browser the user is using. It has been demonstrated [30] that by employing CSS in conjunction with remotely hosted content, it is feasible to find distinctive properties that are specific to different browsers. This method defines CSS properties, such as `background-image`[13], with data hosted remotely – an URL of an image in this case. Because the implementation of CSS may differ across browsers, some CSS properties may not be supported. Different browsers will then send the server requests for different combinations of supported elements. The server that hosts these data can then analyze received requests and determine which browser they came from.

An alternative approach is the utilization of media queries, which specify the width of the user's device and send requests to the server based on the user's screen size. This serves as an additional method for identifying the user [30]. Another potential strategy is the usage of distinct fonts for the rendering of text. Initially, the font would try to be sourced locally. If this were not feasible because the font was absent, a request to the server would be initiated, which could serve as an additional identification method by differentiating the fonts the user has installed.

In contrast to earlier techniques, the deactivation of JavaScript will not impact the viability of this fingerprinting approach. The previously mentioned method of limiting supported fonts described in the previous Font Fingerprinting subsection may diminish the accuracy of this fingerprinting technique. However, it will only render it somewhat ineffective. The sole option would be to prevent the execution of CSS entirely. However, this would inevitably result in the disruption of the majority of web pages' functionality and severely impact user experience.

AudioContext Fingerprinting

AudioContext[14] is part of the Web Audio API[15]. This API controls audio within the web browser, for example, by selecting audio sources or modifying audio properties. Audio-

[13] https://developer.mozilla.org/en-US/docs/Web/CSS/background-image
[14] https://developer.mozilla.org/en-US/docs/Web/API/AudioContext
[15] https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API

Context is used as a graph representation of multiple audio sources and offers methods for manipulating the audio processing.

The methods provided by AudioContext can be utilized to create a unique fingerprint of a given device [14]. This fingerprinting method is possible because each browser has a distinct implementation of AudioContext. A potential fingerprinting method may be based, for instance, on measuring the latency between the output of AudioContext being passed to the system audio.

The most straightforward method for preventing this type of fingerprinting is entirely disabling the Web Audio API. However, this may break some websites that rely on this API for functionality. An alternative approach is similar to the previously described method for mitigating canvas fingerprinting and involves the introduction of random noise into the output [25].

Extension Fingerprinting

The presence of browser extensions can be used as another fingerprinting method. Several methods can be employed in order to confirm whether or not an extension is present. Many browser extensions modify the visited page, for instance, by removing unwanted advertisements or tracking elements. Starov et al. [28] have demonstrated that such alterations themselves can be used as a fingerprint. If an extension removes a Hypertext Markup Language (*HTML*) element, the Document Object Model (*DOM*) is altered. The same is true should an extension add a new element into the DOM to, for example, provide additional functionality. This fingerprinting method is based on the analysis of the DOM. As different users employ different extensions, the changes they introduce to the DOM may serve as a fingerprint, providing further information towards the final identification of the user.

A method of preventing this fingerprinting technique would be randomly changing the DOM structure [28]. However, many users would have to utilize the randomization to be effective. Another downside is that it could result in a decline in user experience.

2.2 Pre-emptive Tracking Prevention

There are multiple methods for blocking tracker functionality. Some of them were already presented in the previous section. One of the most straightforward approaches may be disabling JavaScript, other dynamic browser functionality, or certain HTML elements entirely. Another method is to identify the domains that host the trackers and then block any outgoing HTTP requests directed to them. Another potential solution is to block or modify specific browser functions that the tracker utilizes, thereby preventing it from functioning even if it is downloaded with the page, as shown in the previous Section 2.1.

The methods described in this section work not by blocking specific user identification methods, which were explained in the previous section, but rather by preventing trackers from launching in the first place. However, because of their broadness, they may, in some cases, block more than just trackers or not block some of the trackers at all.

A page can contain elements that may include trackers even if the page itself does not contain any. These elements may originate from third-party sites and take the form of, for example, advertisements.

2.2.1 Domain Blocking

For a code loaded with a web page to be executed, it is first necessary to download it. In order to connect to a website, an HTTP request is sent to the server. As a response to the request, the page and all its contents, including executable code, are downloaded.

The domain-blocking method functions by filtering malicious outgoing HTTP requests. If a request is identified as being directed towards a specified domain that is known to contain tracking elements that are deemed undesirable, the request is blocked [27]. Consequently, even when a page loads scripts or other data from a third party, it can be blocked mostly without compromising the original functionality of the page. However, this depends on how critical the requested content is in the context of the page functionality. A page component may contain both desired content and tracking mechanism. Not loading this component would therefore break the intended functionality. Also, if a page loads its content entirely from a third party and the user blocks any requests made to this third party, it would also disrupt the original page.

The utility of this blocking technique lies in its capacity to provide users with a straightforward method of blocking content from known tracking domains. Consequently, it is a technique predominantly employed by ad blockers to prevent the display of unwanted advertisements. However, the same underlying principle can be extended to the blocking of trackers. In many instances, trackers are integrated into advertisements themselves [17], which might suggest that even a basic ad blocker lacking explicit privacy-enhancing functionality may offer enhanced user privacy due to the blocking of trackers.

The process for determining which requests to block or allow is based on a list. While it is possible to employ a *whitelist* approach, whereby only specific sites are permitted while all others are blocked, this would be highly impractical. Consequently, these lists are in the form of a *blacklist*, containing Uniform Resource Locator *URL* addresses or regular expressions to identify requests that are to be blocked.

Existing Blacklists

Given the considerable number of trackers and other unwanted content, capturing them in a single list is not feasible – not only would it hurt performance, but maintaining such a list would be impractical. Consequently, multiple lists have been created. These lists are usually maintained by individuals who often use their intuition or experience when creating them rather than a systematic approach [27]. Therefore, others could find harmless what one person may consider unwanted content. Some lists are focused on a specific subset of unwanted domains, for example, domains in particular countries, while others are more general, containing multiple unrelated domains. An example of such a general list is *EasyList*[16] or *EasyPrivacy*[16]. While *EasyList* is more targeted towards blocking advertisements and other such unwanted content, *EasyPrivacy* is explicitly oriented towards blocking trackers. These lists can become quite extensive – as of October 2024, *EasyList* has over *71 000* entries.

Nevertheless, the implementation of address filtering has its own set of challenges. In order to best utilize domain blocking, it is essential to have a comprehensive understanding of the specific pages and domains that should be blocked. Otherwise, even valid non-tracking domains might be blocked. However, creating a list of all domains containing undesirable content is difficult. Consequently, if a domain contains a tracker but is not

[16] https://easylist.to/

included on the list, it will not be blocked. Because of that, it is essential to update the list regularly. Furthermore, if a tracker or other unwanted content changes its domain, the list must be updated.

2.2.2 Element and JavaScript Blocking

As mentioned at the beginning of this section, even if a page does not contain any trackers, they may still be present because of elements such as *iframes* or others. This third-party tracking can be mitigated by blocking such HTML elements, thus preventing any code inside them from executing.

Trackers also use JavaScript and other dynamic browser functionalities, as was shown in Subsection 2.1.3. However, differentiating between a tracker and a legitimate code is complicated. One way to solve this problem is to block all JavaScript [27] and other such dynamic browser functionality. Blocking all JavaScript could, however, pose a problem because according to a survey[17], as of October 2024, *98.8%* of all pages available on the Internet use JavaScript in some way. Thus, blocking all JavaScript code would probably impact functionality on many sites, rendering many of them unusable.

2.3 Anti-tracking Tools

This section will present a selection of existing tools that can be used to block fingerprinting methods described in previous Sections 2.1 and 2.2. These tools will be in the form of both browser extensions and browsers with natively implemented anti-fingerprinting measures.

2.3.1 Browsers

The browsers presented in this section all offer a variety of privacy-enhancing measures by default. Consequently, installing additional privacy-oriented extensions is not as crucial as in different browsers that lack native anti-tracking measures. All the presented browsers are free of charge, meaning there is no cost associated with using them. However, some may offer additional functionality for a payment.

Brave

Brave[18] is a privacy-oriented browser that natively supports various anti-tracking and ad-blocking techniques. The browser is developed by Brave Software Inc., a privately held company. *Brave* is open-source and based on the Chromium[19] browser, which is itself open-source. While Brave is free by default, it also offers a paid built-in VPN. The browser's default settings block trackers, third-party cookies, and fingerprinting techniques. Additionally, the browser can be configured to block JavaScript, upgrade HTTP connections to HTTPS when available, or block all cookies.

To prevent the display of advertisements and the tracking of user activity, *Brave* employs a domain-blocking method described in Section 2.2.1. Furthermore, *Brave* randomizes browser API calls to prevent the functioning of specific API-based fingerprinting techniques, such as canvas fingerprinting or WebGL fingerprinting, as previously discussed in Section 2.1.

[17] https://w3techs.com/technologies/details/cp-javascript
[18] https://brave.com/
[19] https://www.chromium.org/

Avast Secure Browser

Avast Secure Browser[20] is a browser that prioritizes the protection of user privacy and security. It is a proprietary product developed by Gen Digital Inc., utilizing the Chromium browser as its core. *Avast Secure Browser* is provided free of charge, while a paid *Avast Secure Browser Pro* version is also available, offering improved functionality such as a built-in VPN. It incorporates ad-blocking and anti-tracking capabilities and also includes an extension guard, which prevents the installation of untrusted extensions. *Avast Secure Browser* also uses domain blocking. One of the filtering lists used is *EasyList*. Furthermore, it performs randomization and generalizing of API responses for elements such as canvas or the navigator interface.

Mozilla Firefox

Mozilla Corporation, a subsidiary of the non-profit Mozilla Foundation, is responsible for the development of the *Mozilla Firefox*[21] browser. The software is open-source, with the code available online. It is based on the Gecko engine, which Mozilla Foundation also develops. It offers third-party cookie blocking, crypto-mining software blocking, and fingerprinting protection by default. The fingerprinting protection is based on blocking requests to third parties and altering specific attributes.

Tor Browser

Tor Browser[22] is maintained and developed by the non-profit Tor Project. It is open-source. It is based on *Mozilla Firefox* but is much more anonymity and privacy-oriented. It has been designed to run with *Tor*, a program that redirects all traffic through a distributed network of relay servers. On each server, all data has encryption added or removed, which makes it much harder for the destination site to know who is communicating with it. However, due to the complexity of Tor network routing, it may load web pages slower than other browsers.

It is advised that no additional extensions should be installed, as they may have a negative impact on the privacy-preserving techniques employed by *Tor Browser*. Upon termination of a session (meaning when the browser is closed or the session is manually renewed), all cookies are automatically deleted. Furthermore, anti-fingerprinting techniques are utilized. The objective of *Tor Browser* is to reduce the uniqueness of each user. For instance, it does not launch in full-screen mode but rather in a window of a predefined size, thereby rendering screen resolution fingerprinting much less effective.

2.3.2 Extensions

Many privacy-oriented extensions are currently available for download. For the purposes of this thesis, a selection of these has been chosen for comparison. As of October 12th, 2024, all selected extensions are available for download from the Chrome Web Store and the Mozilla Addons page. All the extensions described are free to download, although some may offer paid upgrades.

[20] https://www.avast.com/secure-browser
[21] https://www.mozilla.org/firefox/
[22] https://www.torproject.org/

uBlock Origin

uBlock Origin[23], developed by Raymond Hill, is a content-blocking extension. It is an open-source software. It employs the domain blocking method, with numerous pre-configured filter lists, including *EasyList* and *EasyPrivacy*. Consequently, it can block advertisements, trackers, pop-ups, and other undesired content.

It will soon be removed from the Chrome Web Store and cease to be functional on Chromium-based browsers due to its failure to comply with the latest Manifest v3[24] rules as set out by Google. A Manifest v3-compliant version, *uBlock Origin Lite*, is a replacement with limited functionality. However, the original will remain available on *Mozilla Firefox*.

Privacy Badger

Privacy Badger[25] *Privacy Badger* is designed to block advertising and third-party trackers. It is open-source software developed by the non-profit organization Electronic Frontier Foundation. The software is primarily designed to block trackers and does not interfere with advertisements unless they contain trackers. The software is provided with a set of pre-existing lists of trackers, which will be blocked by default. This list can be extended algorithmically through the analysis of trackers, and only when a tracker is identified as such will it be blocked. Furthermore, it obstructs using third-party cookies, canvas fingerprinting, and super cookies.

Adblock Plus

Adblock Plus[26] Its primary function is to facilitate the blocking of advertisements. A premium version is available for purchase and offers additional features. It is open-source and developed by Eyeo GmbH. It employs the domain-blocking method, which may offer limited tracker-blocking functionality as was described in Section 2.2.1, thus partially enhancing the user's privacy. However, *Adblock Plus* does not block all ads. By default, the developers have chosen some *acceptable* ads that are allowed.

JShelter

JShelter[27] is a privacy-oriented extension developed by Libor Polčák and Giorgio Maone. By default, *JShelter* provides anti-fingerprinting methods, such as randomizing the canvas fingerprint. Additionally, the software employs a heuristic monitoring system to detect the use of browser APIs commonly associated with fingerprinting techniques. Upon detecting such fingerprinting activity on a given website, a notification is generated and displayed on the user's computer. *JShelter* can be configured to prevent further HTTP requests to such websites. However, Manifest v3 restricts its functionality on Chromium-based browsers.

[23] https://ublockorigin.com/
[24] https://github.com/uBlockOrigin/uBlock-issues/issues/338
[25] https://privacybadger.org/
[26] https://adblockplus.org/
[27] https://jshelter.org/

Ghostery

Ghostery[28] is an extension designed to prevent tracking and block advertisements. Ghostery GmbH developes the software. The code is open-source. *Ghostery* can block advertisements, cookies, trackers, and other elements, including various pop-ups.

NoScript

NoScript[29] is natively part of the *Tor Browser*. It is an open-source software developed by Giorgio Maone. Unlike the previously described extensions, it is not designed for subtle anti-tracking purposes such as randomization or generalization of API responses, neither does it block HTTP requests. Instead, it pre-emptively blocks JavaScript and other potentially unwanted content.

[28] https://www.ghostery.com/
[29] https://noscript.net/

Chapter 3

Existing Solutions for Testing Anti-tracking Tools

To enable fair and meaningful comparisons of anti-tracking tools, it is first necessary to set up a stable and easily replicable testing environment. Such an environment should include an automated system that minimizes human interaction and allows for fair and repeatable evaluations of the anti-tracking capabilities of different browsers and extensions.

Since trackers are present on web pages accessed through the browser, the environment must also replicate user activity, such as visiting the tracking sites. Subsequently, specific web pages implementing various tracking methods, described in Chapter 2, can be accessed to compare the uniqueness and stability of generated fingerprints. This chapter presents the reader with an overview of existing tools used to compare the anti-tracking capabilities of a given tool.

3.1 Environment

The purpose of a specific environment is to make results easily replicable. That would mean that conditions under which testing has occurred can be replicated across multiple systems. This section describes two possible approaches – *virtualization* and *containers*.

Virtualization

One of the ways to create a replicable environment is by using virtualization [2]. An example of virtualization software is *VirtualBox*[1]. Virtualization allows multiple virtual machines to run concurrently on a single physical machine. Virtual machines are emulated physical computers that do not need their own hardware to run but instead rely on the hardware of the host machines. Host hardware can be shared among multiple virtual machines.

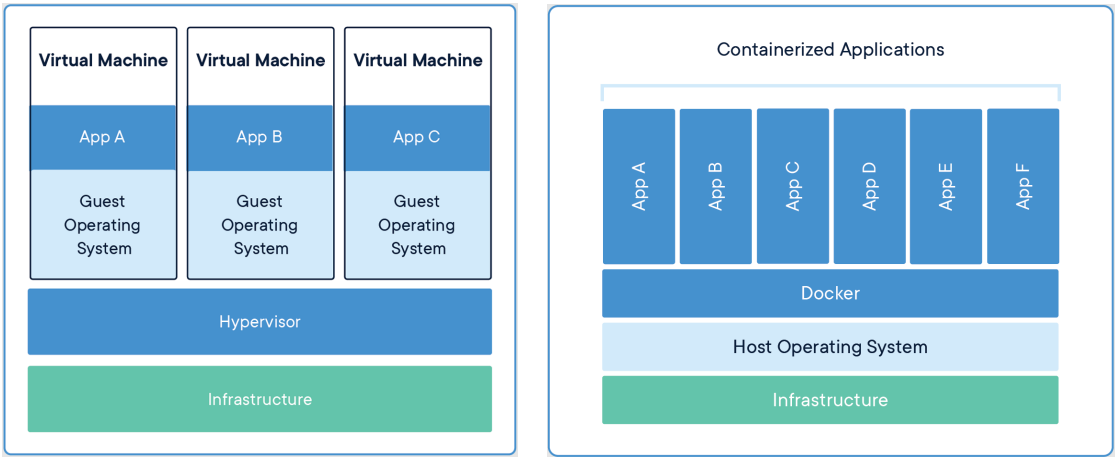
Virtual machines are controlled by a *hypervisor* – software designed to provide a level of abstraction from the host system. It also manages the resources allocated to each virtual machine and prevents virtual machines from affecting each other by isolating them. The virtualization process is visualized in Figure 3.1a. Each virtual machine can run its own operating system, such as Linux or Windows. Other than that, each virtual machine can be configured for specific purposes, such as running a web server or, for the purposes of this

[1] https://www.virtualbox.org/

thesis, a comparison of anti-tracking tools. While virtualization is a viable approach, using a whole virtual operating system may be unnecessarily complex for certain purposes.

Containers

Another method for environment creation is using containers [29]. An example of container software is *Docker*[2]. Containers are software that can be executed to run an application (or multiple applications) they contain. The code and configuration of each application are included in the container, together with libraries the application uses and other required dependencies. Containers rely on operating system-level virtualization, isolating processes from each other and controlling what resources each process can access. Container technology is showcased in Figure 3.1b. A practical difference from the previously mentioned virtualization approach is that containers do not need their own operating system. Instead, they rely on the host computer's operating system, which makes them less isolated than virtual machines but offers better performance. Relying on the host operating system makes them more lightweight in terms of both actual size and the overhead, which is less than with virtualization.



(a) Illustration of the virtualization technology

(b) Illustration of the container technology

Figure 3.1: Illustration of virtualization and containers[3]

3.2 Existing Tools and Methods

This section presents several tools and methods for testing different anti-tracking technologies. These tools and methods often rely on browser automation. Automating interaction with the browser itself and different pages on the internet allows programmers to simulate the activity of real users. This activity may include accessing a page or interacting with HTML elements on the visited page. These interactions are crucial to activating the tracker and measuring whether or not it has been blocked or if the calculated fingerprint is unique

[2] https://www.docker.com/

[3] taken from https://www.docker.com/resources/what-container/, visited 2024-11-04

and stable. An example of a tool that is frequently employed by specific anti-tracking measurement tools for browser automation is *Selenium*[4].

Straightforward approach

The first presented method is not a standalone tool but a possible method for benchmarking anti-tracking browser extensions, proposed and implemented by Traverso et al. [31] in 2017. Their method only compares anti-tracking efficiency in terms of the number of known tracker domains contacted. Their setup uses Selenium to simulate the *Mozilla Firefox* browser. A list of URLs is used to specify which pages to visit. Each page is visited 10 times, with the browser cache cleared after each visit. A different configuration is used for each crawl, consisting of different privacy-oriented extensions manually installed in the browser. HTTP communication between the site and the client on each page visited is stored and analyzed later. This communication includes, among other things, bytes downloaded, time to load the page, tracking domains, and contact with third parties. The statistics resulting from this data are later used to compare each extension. The tool continues with another page after a page has been successfully loaded (including all trackers), indicated by the JavaScript *OnLoad* event. If a page visit is unsuccessful, the page is ignored.

Open-WPM

Open-WPM [12] was released in 2016 but is still updated as of 2024. It is open source, with its code available online[5]. The main purpose of this tool is to simulate the user's activity and record data from the simulated interactions. This data may include cookies, response metadata, or script behavior. This indicates that the tool can observe stateful and stateless tracking methods. In addition, OpenWPM can store the state of the browser over multiple measurements, allowing it to observe tracking techniques such as cookie respawning, which is the basis of supercookies. OpenWPM consists of three modules – multiple *browser managers*, *task manager*, and *data aggregator*. The system's design is shown in Figure 3.2.

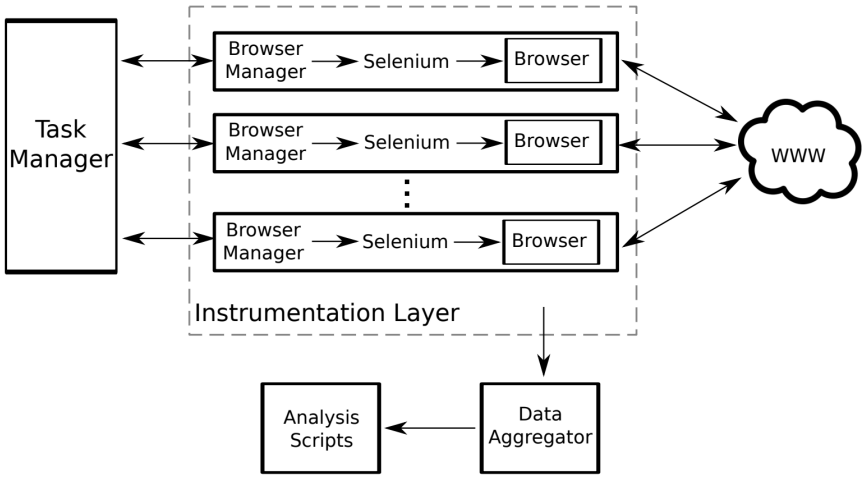


Figure 3.2: High level design of OpenWPM, taken from [12].

[4] https://www.selenium.dev/
[5] https://github.com/openwpm/OpenWPM

Browser managers are used to run separate instances of full-fledged *Mozilla Firefox* web browsers via Selenium. Managers are run in parallel to improve performance. When a new browser manager is created, a specified configuration is applied. This allows each session to use a different configuration, such as a different combination of browser extensions, which may themselves be further configured. In addition, each manager can be configured to perform different tasks. These tasks may consist of, for example, visiting different websites. These managers can also be launched in headless mode, using a virtual frame buffer for the graphical interface without displaying it to the user.

Task manager is used to control multiple instances of browser managers. Commands for each browser are stored in an execution thread unique to each browser. If the browser manager encounters an error, such as a timeout or crash, the current state of the browser is archived. The archived state allows the browser to be restarted later in the same state (e.g., cookies stored from visited sites).

Data aggregator is used to store data received from browser managers. It stores relational data and compressed web content, which includes all types of cookies (such as regular cookies or Flash cookies on legacy systems), HTTP (and HTTPS) request and response headers, and finally, access to various JavaScript prototypes such as storage, canvas, or navigator. The data is stored in a structured way, allowing it to be traced back to a specific browser and page from which it originated. Upon completion, this data can be analyzed to determine, for example, which extension blocked the most tracking attempts.

PETInspector & FPInspector

PETInspector and *FPInspector* [8] are used together to form a benchmarking system, where *FPInspector* evaluates the results provided by *PETInspector*. *PET* stands for *Privacy Enhancing Technology*, and *FP* stands for *Fingerprint*. It was developed in 2019 but has not been updated since. *PETInspector* is available on GitHub[6]. The system can be used to compare different privacy-enhancing extensions and browsers. *PETInspector* serves as a tool to experimentally calculate the anti-fingerprinting capabilities of a given privacy-enhancing tool, which then forms a mask of how the tool behaves. This mask is then used with actual observed fingerprints to simulate how a browser would behave if everyone were using the given privacy-enhancing tool. The resulting traceability is then measured, which is the purpose of the *FPInspector*. This experimental and observational approach forms the hybrid benchmarking system. The output of the system is the effectiveness of a given tool.

The system consists of four parts. These parts are the *client simulator*, *fingerprinting server* and *analysis engine*, which form the *PETInspector*, and the *FPInspector*, which is the fourth part. This is illustrated in the Figure 3.3.

Client simulator simulates users via Selenium. Each simulated user has various privacy-enhancing tools installed. As mentioned above, these tools include browsers and extensions with anti-tracking capabilities. These simulated users then visit the fingerprinting server. The visits consist of accessing different domains, reloading a domain, and browsing multiple sessions. The simulator also allows the configuration of different fonts, time zones, and other properties.

The fingerprinting server uses various fingerprinting techniques to compute the fingerprints of visiting clients.

The analysis engine uses information from both the client simulator and the fingerprinting server to measure how a particular client configuration affects a fingerprint by observing

[6] https://github.com/tadatitam/pet-inspector

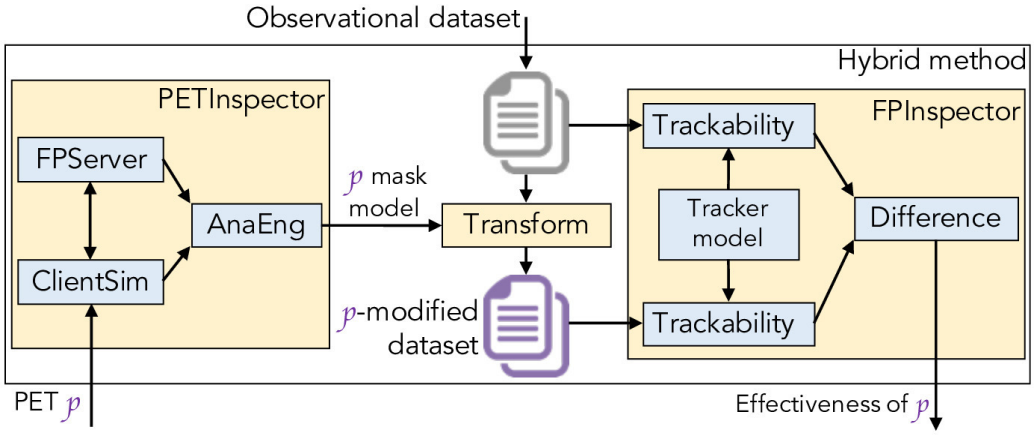


Figure 3.3: Benchmarking system design showcasing how *PETInspector* and *FPInspector* interact, taken from [8].

whether the data the browser passed to the fingerprinting server was changed by generalization or randomization. If the result of the fingerprinted property is changed by the privacy-enhancing tool, either by generalization or randomization, the property is modeled as masked in the generated mask and unmasked if not changed. This approach overestimates the effectiveness because it treats any change to the property values as unrecoverable, rendering the property useless for tracking.

FPInspector receives two fingerprint datasets as input. The first dataset consists of actual measured fingerprints with no privacy-enhancing tool installed. The second consists of simulated fingerprints showing what the first dataset would look like if everyone used a particular privacy-enhancing tool. The masks created by *PETInspector* are used to obtain the second data set. These datasets are then compared for uniqueness to calculate effectiveness.

Browser Extension Testing Environment by Jana Petráňová

The Browser Extension Testing Environment [22] has built-in methods for testing privacy-oriented browser extensions. It does not directly compute the effectiveness of a given privacy-oriented tool but returns information about how it affects tracking. It has no official name. The environment was proposed as part of the master's thesis by Jana Petráňová [22]. While the environment is mainly designed to test the *JShelter* extension, it also supports testing other privacy-oriented extensions. The extensions are tested in two currently supported browsers: *Google Chrome* and *Mozilla Firefox*. It uses the designs and implementations proposed by *Open-WPM* and *PETInspector* described above. The environment consists of two main parts – *integration testing* and *system testing*. Its high-level design, together with the technologies used, is shown in Figure 3.4.

Integration testing is based on the PETInspector tool. It simulates a client using Selenium. The client has either a specific privacy-oriented extension or multiple extensions installed. The client then visits a site hosted by a local server. The visited sites can be further configured, but only one local site is provided by default. The server uses several customizable tracking methods to calculate the client's fingerprint. The attributes observed by the trackers are stored. They can be analyzed later to understand how a given extension affects the results returned by various browser methods.

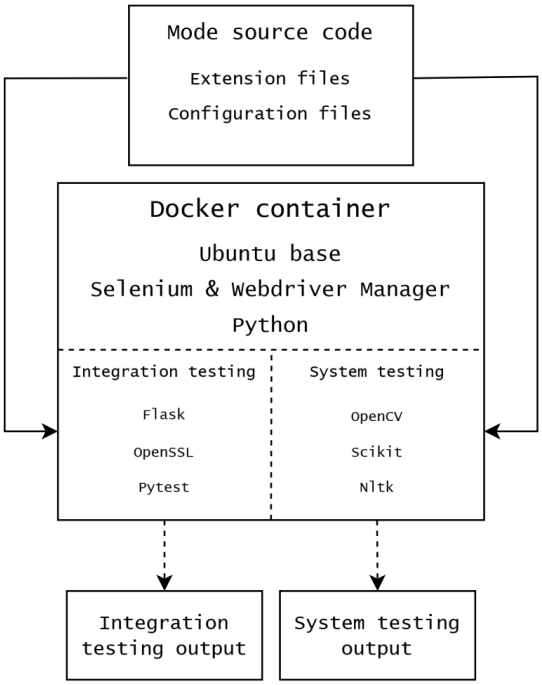


Figure 3.4: Architecture of the testing environment for anti-tracking tools, taken from [22].

System testing measures how a specific combination of installed privacy-oriented extensions affects visited pages. Again, a client is simulated using Selenium. The client then visits many specified web pages to measure how the extension(s) affect the page content. Each visit is logged. The logs can then be analyzed to understand how an extension affected the visited page. The logs include the similarity between the console output of the unaffected page and the console output of the same page with an extension installed. Also, a screenshot of the visited page is taken before and after the extension is installed. The images are automatically compared to measure if and how an extension has changed the visual layout of the page.

3.3 Fingerprinting Tools and Sites

Many sites already exist that calculate a browser fingerprint and present the result to the user. Some even measure and inform how traceable the browser visiting the page is by measuring the similarity of observed properties to other records in their database. These tools can be used to compare how unique a browser is or if the fingerprint changes over multiple visits or browser sessions.

AmIUnique

AmIUnique[7] is one such site. The site's purpose is to build a fingerprint database for research purposes. It uses both active and passive fingerprinting techniques. It informs the user how unique their browser is among other fingerprints in the database. This uniqueness is further broken down into each property tracked – users can see how many other users

[7] https://amiunique.org/

share the results of each measurement, such as screen resolution, installed fonts, or canvas display.

PrivacyCheck

PrivacyCheck[8] is another site that allows users to fingerprint their browser using many different techniques, both active and passive. The site was implemented for research purposes, similar to Am I Unique. Each fingerprinting method collects the necessary data and informs the user of the calculated fingerprint. The site can also measure the stability of fingerprints over multiple visits.

Browserleaks

Browserleaks[9] demonstrates several fingerprinting methods, both active and passive, which then provide the calculated fingerprint to the user. Some methods even inform the user of the uniqueness of his fingerprint, making it possible to measure the effectiveness of both randomization and generalization anti-fingerprinting approaches. Other methods only inform the user of the calculated fingerprint, making them useful only for measuring the effectiveness of randomization.

NoScriptFingerprint

NoScriptFingerprint[10] is different from the previous pages in that it calculates the browser fingerprint entirely without using JavaScript. It still uses active and passive fingerprinting techniques, but instead of JavaScript, it relies entirely on other technologies, such as CSS, to calculate the fingerprint. The site informs the user of their calculated fingerprint and the value returned for each tracking property, which again is only applicable for the randomization anti-fingerprinting approach. Unlike the previous pages used for academic purposes, this page is hosted by FingerprintJS, Inc.

Supercookie

Supercookie[11] uses *favicons* to compute the fingerprint. Since the *favicon* cache is not often cleared by users, it can be used to store a persistent fingerprint. It is loosely similar to an evercookie, as described in the Subsection 2.1.2, since it also uses an uncommon mechanism to store the identifier, although the identifier is saved only in this single storage. However, it is quite difficult to delete. Because of how it works, there is no point in measuring fingerprint uniqueness since no identifier will ever be used twice. The reason is that for each unknown visitor, a new identifier is created on the server. However, it can be used to observe whether an anti-tracking tool can subvert this tracking method by checking whether the calculated fingerprint changes between sessions.

[8] https://privacycheck.sec.lrz.de/
[9] https://browserleaks.com/
[10] https://noscriptfingerprint.com/
[11] https://supercookie.me, repository available at https://github.com/jonasstrehle/supercookie

Chapter 4

Environment and Comparison Design

As previously outlined in Chapter 2, multiple methods for user tracking exist. Consequently, privacy preserving tools that prevent the tracking have been developed. However, these tools are mainly independent of each other, and thus differ in both the anti-tracking methods they provide and their effectiveness. While some tools, such as JShelter[1], have explicitly stated that they should be used together with other different anti-tracking tools, other tools should be used with no additional tools present, such as the Tor Browser.

Since not all anti-tracking tools have their source code publicly available, comparing their functionality by studying the code is impossible. Hence, specialized environments and tools have been developed for the purpose of comparing the effectiveness of the anti-tracking tools. A number of these tools have been described in Chapter 3. The main objective of this thesis is to improve existing comparisons of anti-tracking tools. To design an environment for the comparisons, it is first necessary to establish the requirements. The basic requirements are summarized in the following list:

- The environment should be documented and easy to replicate.
- The environment should support multiple anti-tracking tools.
- The environment should support the addition of future fingerprinting methods.
- Comparison methods should be clearly defined, accompanied by their respective limitations.
- Comparison methods should be deterministic, making the results replicable.

In order to address the specified requirements, this chapter presents a testing environment in which the comparisons will be performed. Furthermore, this chapter discusses potential comparison methods and identifies their limitations.

4.1 Environment for Comparisons

One of the requirements for the environment is to ensure its replicability, which can be accomplished by employing the technologies presented in Section 3.1. Furthermore, the en-

[1] https://jshelter.org/faq/#what-other-extension-do-you-recommend-to-run-along-jshelter

vironment should support multiple anti-tracking tools, including existing tools, as outlined in Section 2.3, and additional tools that may emerge in the future.

As tools and environments to compare anti-tracking tools already exist, this thesis will use them. Some of these tools were described in Section 3.2. *OpenWPM* is the tool often used in existing anti-tracking research [21, 15, 18]. While *OpenWPM* was explicitly designed for web privacy research and is still updated even in 2024, it lacks one crucial feature: the support of other browsers. *OpenWPM* can only be used with the *Mozilla Firefox* browser, which is insufficient for this thesis. This thesis aims to compare the anti-tracking performance of different browsers, as discussed in Section 2.3. Alternative comparison tools are usually outdated or incompatible with the stated requirements.

However, the existing *Browser Extension Testing Environment by Jana Petráňová* fulfills the requirements for an environment. It already supports two major browsers: *Google Chrome* and *Mozilla Firefox*. Both *Chromium*, which is used by *Google Chrome*, and *Mozilla Firefox* are often used as the base for other browsers. For instance, *Tor Browser* is based on *Mozilla Firefox*, and *Brave* is based on *Chromium*. Consequently, the environment could be extended to support additional privacy-preserving browsers.

The environment is configured to support multiple anti-tracking browser extensions, and the list of supported extensions can be extended. All other parts of the environment are similarly configurable.

For this thesis, the environment will be extended to support additional browsers and extensions. Integration and system testing will serve as the basis for the comparisons. The comparison will use some of the implemented test methods, as well as new test methods that will be explained in the next section.

4.2 Comparison Methods

The existing environment consists of two parts: *integration testing* and *system testing*. Integration testing can be described as evaluating the functionality of a given privacy-preserving tool. This evaluation involves comparing the results received with and without the presence of the anti-tracking tool to ascertain whether the tool alters API responses. System testing involves evaluating the impact of the anti-tracking tool on actual web pages by comparing the *blank* state of the page, defined as the page's appearance and behavior in the absence of the anti-tracking tool, with its state with the tool present.

This thesis employs and extends both existing parts to evaluate and compare privacy-preserving tools in several ways to answer the three following questions.

- How many and what tracking methods are prevented?
- How much have the API responses changed?
- How are real sites affected by preventing the tracking?

There are many approaches to evaluate tracking prevention. However, no comparison can fully answer which tool is superior to another. Therefore, while the obtained results may still be valuable, it is important to understand what the limitations of each comparison method are.

4.2.1 Evaluation of API Response Changes

The first comparison method establishes the foundation for subsequent methods. It evaluates the efficacy of the privacy-preserving tool in modifying the API responses utilized for fingerprinting, quantifying the number of modifications made. For each fingerprinting method, the outcome will include information regarding the alteration of the fingerprinting API result and the number of prevented fingerprinting attempts. An overview of the tracking methods employed for this comparison can be found in Chapter 2. The environment already contains a fingerprinting server that utilizes various technologies to calculate a fingerprint. Initially, a default *clean* state is obtained through a client visit with a selected browser, with no privacy-preserving tools in place. Subsequently, the same visit is repeated with a selected tool (or multiple tools) present. The results of each visit are compared with predefined values, obtained from observing behavior of a given tool. For the purposes of this thesis, the result of the *clean* state will be calculated only once. The *clean* results will then be compared with those of a given anti-tracking tool to understand which APIs were changed by the tool. This process is visualized in Figure 4.1.

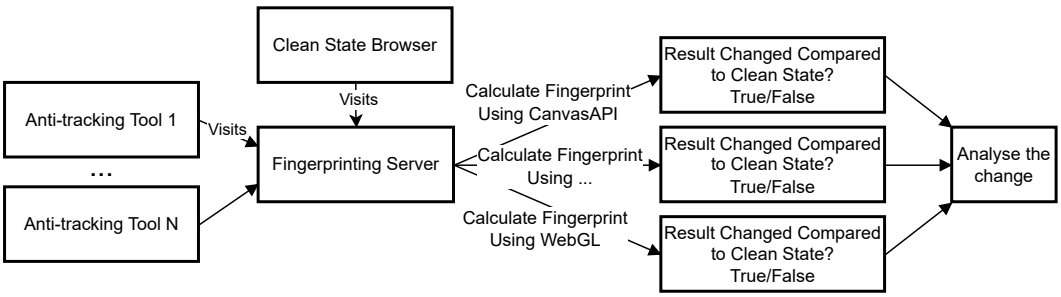


Figure 4.1: Fingerprinting server calculates a fingerprint using multiple browser APIs. Afterwards, the obtained results are compared against the clean state to understand what APIs has the given anti-tracking tool affected. *Anti-tracking Tool* includes both anti-tracking extensions and browsers with built-in anti-tracking capabilities. *Clean State Browser* denotes browser with no anti-tracking capabilites or extensions.

Additionally, some tracking prevention techniques described in Chapter 2 have been shown to have a negative impact on user browsing comfort. In order to understand the changes made to prevent fingerprinting, the environment will be expanded to include an analysis of the changed API results. A given tool may only modify the API result by flipping the value of a single bit. While this may result in a difference, the change can be easily reversed. Thus, for selected fingerprinting methods, such as Canvas fingerprinting, this comparison will observe how the calculated fingerprint has changed regarding which pixels were affected and how much. This will give the user another way to understand how a given tool operates.

The integration tests were primarily designed to test the JShelter extension. Additional fingerprinting methods will be incorporated to broaden the scope of the comparisons. These novel methods will be based on Cookies and other existing fingerprinting implementations, as outlined in Section 3.3.

The outcomes of this comparison can be utilized to determine the efficacy of a given tool in preventing fingerprinting attempts. If one privacy-preserving tool prevents more

fingerprinting attempts than another, it may indicate that the first tool is superior in blocking multiple fingerprinting.

Limitations

A difference in the number of changed API responses does not necessarily indicate that one tool is more effective than another in preventing fingerprinting. For instance, a tool may not have flagged a fingerprinting attempt as harmful due to its execution in a controlled environment despite its potential efficacy in a real-world setting. Additionally, it is possible that a tool did not prevent the tested fingerprinting methods but may impede numerous other untested fingerprinting attempts. It is also important to note that specific tools are designed to have a limited scope, and thus, they may only obstruct specific fingerprinting techniques. This is considered their intended behavior and not an indication of worse performance.

The next limitation of this approach is that it does not fully consider the quality of randomizing the API response, as previously mentioned. Even a single bit change can result in a different outcome. Even though this comparison analyzes the changes to several API results, it only illustrates how a given tool affects the APIs, not how difficult reversing them would be.

Despite the limitations mentioned, with them known and understood, this comparison still provides data about how the privacy-preserving tools work and basic information about the scope of their anti-tracking protection. It will also be used as a basis for other more substantial comparisons.

4.2.2 Evaluation of Fingerprinting Prevented by API Response Changes

This evaluation approximates how a privacy-preserving tool behaves in real-world scenarios. Firstly, fingerprinting attempts are observed on real sites. The observed techniques used to obtain the fingerprint are then stored. The stored results are then compared to the results of the previous evaluation, described in the previous Subsection 4.2.1. Since the first evaluation results say what fingerprinting APIs the tool changes, we can approximate what and how many fingerprinting attempts would have been prevented had the tool been present in the browser. This evaluation only applies to tools that prevent fingerprinting by modifying the API responses, either by randomizing or standardizing them. A high-level simplified fictional example of this approach is illustrated in Figure 4.2.

Many anti-tracking tools offer statistics regarding the number of tracking attempts that have been successfully prevented or the number of domains that have been blocked. However, the reliability of these statistics is limited due to the potential inaccuracies in the data and the utilization of different calculation methods by each tool. Consequently, the results obtained from these tools are unsuitable for straightforward comparison. Therefore, it is necessary to employ an alternative method of assessing the performance of these tools.

It is possible to achieve this objective by employing a custom tool to observe tracking attempts. In order to make this observation, the JShelter extension will be utilized, as it contains a Fingerprinting Detector based on Marek Saloň's master's thesis [24], which can be used to observe the actual fingerprinting methods employed on different sites.

During the observation period, no anti-tracking tools will be present, and the JShelter extension will have its anti-tracking functions disabled. Subsequently, the observed fingerprinting attempts can be compared with the known capabilities of a given API-based anti-tracking tool. The previous evaluation will test for all fingerprintable data the Fin-

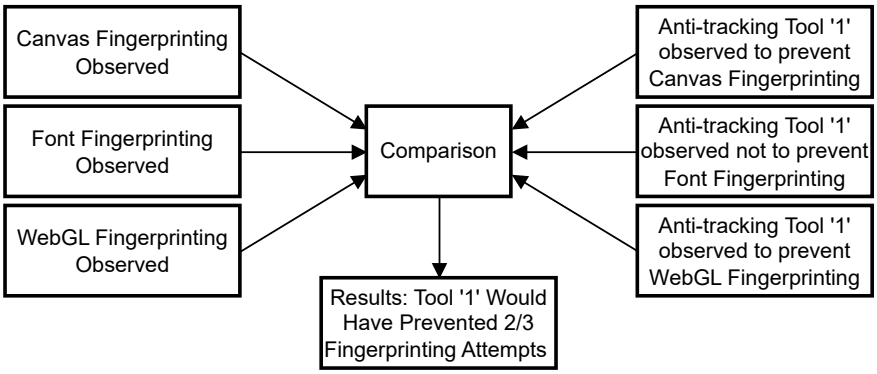


Figure 4.2: Visualization of which of the observed fingerprinting attempts would have been prevented by a given API-based anti-tracking tool.

gerprint Detector may observe. Afterward, it is possible to determine what and how many attempts a given tool would have prevented had it been present in the browser.

However, given the inherently dynamic nature of the web, page content may change between visits. Visiting a given real-world site multiple times to observe the fingerprinting methods used and comparing it to the capabilities of a given anti-tracking extension each time is thus not feasible.

To illustrate, consider a page containing advertisements from third parties. Upon initial visit, specific advertisements are visible. However, these advertisements may have changed upon subsequent visits, which may indicate that different third-party sites hosting the advertisements have been contacted. Since advertisements frequently incorporate tracking mechanisms, these mechanisms may differ between visits.

Therefore, it is important to note that while one tool may have been observed to block all attempts observed on a given site, on the next visit, another tool with the same capabilities may be presented with different fingerprinting methods that neither this nor the previous tool can prevent. Since neither of them can block those attempts, the tool would have much worse results than the previous one, even though they have the same capabilities.

The mentioned problem of unpredictable results can be partially solved by using a larger sample of data. Mazel et al. have used this approach in their comparison [18]. For example, the page could be visited not once, but ten times and the number of prevented fingerprinting attempts may be evened out using an average value. While this method could improve the results, it does not completely solve the problem of different tracking techniques appearing for different extensions over time.

The proposed solution to this issue is to incorporate a measure of determinism by preserving the observed fingerprinting attempts. For each visited page, the corresponding observed fingerprinting methods will be recorded. Subsequently, given the existing profile of each anti-tracking tool, which contains information about the methods it prevents, a comparison can be made between the two. This approach ensures that each anti-tracking tool is compared with the same data, generating results that are more aligned with reality. The full illustration of this evaluation can be seen in the following Figure 4.3.

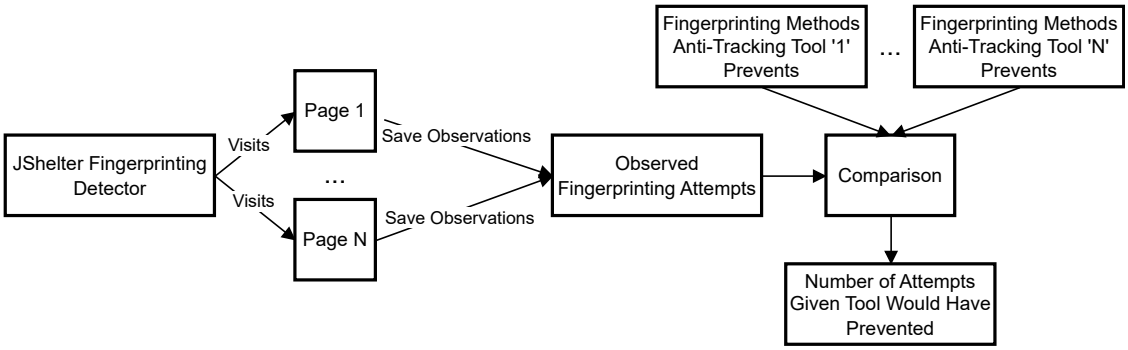


Figure 4.3: Visualization of which of the observed fingerprinting attempts would have been prevented by a given API-based anti-tracking tool. All tools are compared against the same sample of observed fingerprinting attempts. The evaluation is repeated for each anti-tracking tool from the previous comparison.

Limitations

The limitations of this comparison are the same as of the previous method. However, an additional limitation arises from how the detector observes fingerprinting attempts. The primary issue is that it does not observe all fingerprinting attempts but only those for which it has been configured. Consequently, while an anti-tracking tool may block the undetected attempts, and no other tool can, since it will not be detected, it will not be reflected in the results.

A further limitation is that not all APIs utilized for fingerprinting are employed in this manner, potentially leading to the detection of false positives by the fingerprinting detector. Detecting false positives would increase the number of tracking attempts reported and prevented due to the adjustment of results.

4.2.3 Evaluation of Attempts Prevented by Blocking the Source

The previous evaluation method was exclusively for fingerprinting prevention based on modifying the API responses that can be exploited for fingerprinting. This evaluation will utilize a deterministic approach similar to the previous evaluation. However, it can be employed for anti-tracking tools that block requests to specific addresses based on the domain name, URL or other criteria.

As in the preceding evaluation, the reliability of the blocking statistics reported by the anti-tracking tools is questionable. This underscores the need to develop a custom tool capable of observing network traffic associated with a given page, thereby identifying the contacted domains, including those with trackers. However, a caveat persists from the previous evaluation: visiting the same page twice may result in different domains being contacted. Consequently, comparing the number of outgoing connections between visits is not viable. While it may offer a basic understanding of changes in the number of contacted domains across multiple sites, it still does not allow for a fair comparison.

The present issue will be resolved consistent with the previous evaluation, employing the same deterministic approach. To this end, multiple pages (or, in some cases, the same page on multiple occasions) will be visited, and each page's network traffic will be saved to approximate real-world performance. Subsequently, based on the obtained dataset, the

requests to each contacted domain will be re-enacted with different domain blocking tools, and the number of blocked requests will be logged.

This approach, however, still presents a problem. The issue is that the outgoing requests may have a tree structure. To illustrate this, consider the following example: A user visits page `A`, which contains third-party hosted tracking content. The third party is referred to as `B`. Thus, connecting to page `A` also connects to page `B`. However, it is important to note that page `B` may also host additional third-party tracking content from different pages, such as `C` and `D`. Consequently, when a user visits page `A`, a request is generated for content from all three third-party tracking sites: `B`, `C`, and `D`.

Should we observe the number of blocked requests, one domain blocking tool may block requests to sites `C` and `D`, resulting in a total of *two* blocked requests. Subsequently, an additional tool would block the request to page `B`, which was utilized as the initial point for requests to both pages `C` and `D`. However, this tool would not block pages `C` and `D`. Consequently, the tool would only be reported to block *one* request. The problem is that, in a real scenario, the second tool would block all three requests to `B`, `C`, and `D`. This outcome is attributable to the fact that the second tool blocked the request to page `B`, which consequently prevented the pages `C` and `D` from being contacted. If the evaluation is based solely on the number of requests made, the second tool may be considered inferior to the first one in this hypothetical scenario.

The issue can be resolved by maintaining the request chain. When creating the dataset that will be used for evaluation, it is necessary to note the predecessor of each request made. Subsequently, the described issue can be resolved by not checking whether each request was blocked individually. Instead, the approach will go through each level of the tree-like request structure. If a request is blocked on a given tree level, no other child nodes representing further domains are contacted because the parent domain will not be requested. Illustration of the domain request chain is illustrated in Figure 4.4.

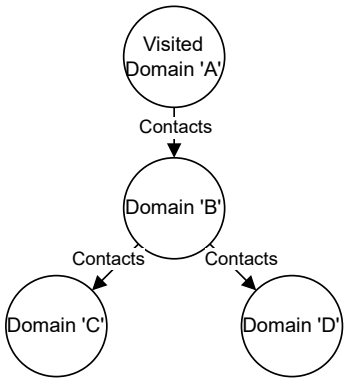


Figure 4.4: Illustration of a tree-like request structure which allows for accurate measurement of number of requests blocked.

Limitations

However, this evaluation is still subject to limitations. As with the previous evaluation, it reports solely on results based on observed requests. Consequently, it is impossible to ascertain how the extension would behave on unvisited pages, which may contain different third-party content. Another limitation is that the lists that serve as the basis of domain blocking are made based on user decisions, as described in Section 2.2.1.

Since people make lists that contain a degree of subjectivity, not all blocked content necessarily deserves blocking. While anti-tracking lists exist, domain-blocking is predominantly employed to block advertisements. This implies that while one domain blocking tool may block more requests than another, the blocked requests may not have contained any harmful tracking software since advertisements may not always contain trackers. Some people might also want to see some advertisements. Consequently, comparing the number of blocked requests offers a limited perspective on the efficacy of a given tool.

In order to gain a more nuanced understanding of the efficacy of a given tool, a manual analysis of each blocked request would be necessary to ascertain the validity of the blocking action. Nevertheless, the results obtained from the described analysis can offer insights into the number of requests blocked and the potential benefits they provide to the user.

4.2.4 Evaluation of Anti-tracking capabilities of Domain Blocking Tools

This evaluation mention intends to quantify the anti-tracking capabilities of a given domain-blocking tool. Specifically, it uses the same tree request structure described in the previous Evaluation 4.2.3. However, it also assigns each address with a number of observed fingerprinting attempts using the same detection mechanism based on the JShelter Fingerprinting Detector as in the Evaluation 4.2.2.

The association of a specific number of fingerprinting attempts with a given domain enables the utilization of the request-chain method previously employed in the domain-blocking comparison. Each node, representing a contacted domain, can be straightforwardly linked to the number of observed fingerprinting attempts. This approach facilitates determining the number of fingerprinting attempts that would not be made without contact with the domain. This methodology enables a direct evaluation of the anti-tracking capabilities of domain-blocking tools, which can then be compared to the API-based tools.

The subsequent example illustrates this methodology. As with the preceding comparison, the domain `A` is contacted, which initiates a request to domain `B`, which subsequently issues requests to domains `C` and `D`. However, let us consider a scenario in which we have associated two fingerprinting attempts with domain `A` and one attempt each with domains `B`, `C`, and `D`. In this scenario, we assume that a domain blocking tool has blocked the request to domain `B`. This would entail the prevention of one fingerprinting attempt by domain `B` and an additional attempt by both domains `C` and `D`. Consequently, the total number of attempts would be reduced by three, leaving only two attempts made by the original page. A subsequent step would involve comparing this number with the number of fingerprinting attempts prevented by randomizing the response, as detailed in the comparison described in Subsection 4.2.2 that focused on them. Illustration of the request tree with associated number of fingerprinting attempts observed can be seen in Figure 4.5.

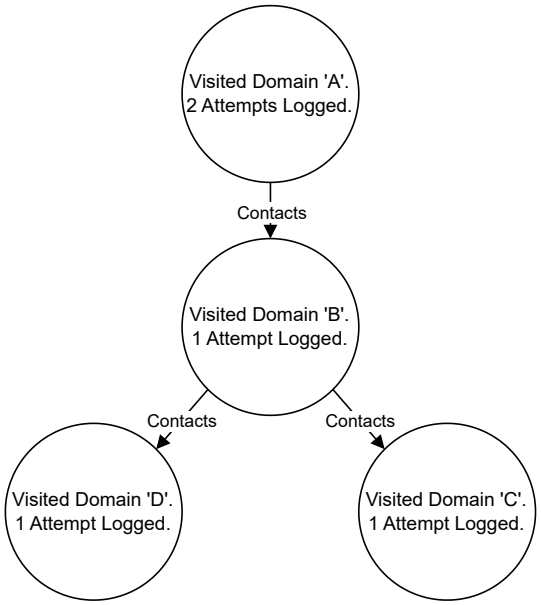


Figure 4.5: Illustration of a request tree with observed number of tracking attempts.

Limitations

The limitations of this approach are derived from the limitations of the previous comparisons on which it is based. Determining whether an observed fingerprinting attempt was utilized for malicious purposes is not feasible. For instance, blocking three attempts by blocking the associated domains does not necessarily indicate superiority over a randomization-based tool, which only prevented one attempt, as those three blocked attempts may not have been actual fingerprinting attempts. Additionally, while domain-blocking tools can prevent third-party content, API-based tools can address first-party tracking. Despite these limitations, this approach offers a method for comparing two distinct methods of preventing fingerprinting.

4.2.5 Evaluation of Effects on Sites

The final comparisons will provide data on how the presence of a particular anti-tracking tool affects actual user experience. The first evaluation will be based on page load time since it is a factor in choosing an anti-tracking tool. Because the tools that modify API results provide overhead, they can increase the time it takes to load a page. Similarly, tools that block domains may decrease the time it takes to load a page because blocking them from loading decreases the time it takes to load the page, even though they also provide some overhead.

Another evaluation will be based on how a given anti-tracking tool changes the visited page. The system testing part of the environment already has an implemented method for measuring visual and functional changes. This thesis will improve the way the visual changes are calculated. Currently, the tests work by comparing the images of the page without any anti-tracking tools with the images with the tools present. However, it is

unusable for the domain-blocking approach since it often blocks advertisements that take up a large part of the page. This thesis will solve this problem by visiting each page multiple times in a short timespan to determine which part changes, which could indicate that it contains advertisements. Afterward, this area would be excluded from comparing images, allowing an automatic comparison with and without advertisements. The process is illustrated in Figure 4.6.

(a) Page *www.novinky.cz* during first visit

(b) Page *www.novinky.cz* during second visit

(c) Detected advertisements area (blurred) based on the difference

Figure 4.6: Visual detection of advertisements during multiple visits of a given page

Limitations

The main limitation of visual comparison is that the ads may not change during the short period between visits. However, as the time between visits increases, news sites, for example, may already be showing new articles that would be marked as part of the unwanted content. While this can be solved using a percentage of the page that would have to change to indicate a breakage, it is still a limitation. Another potential limitation is that some sites may display a *cookie prompt* asking the user if they agree to the page using the *cookies* described in the Section 2.1.1. This prompt can take up the entire page, and since domain blocking tools may remove these prompts, the default page and the page with the anti-tracking tool present can look completely different, limiting this approach. Because of these limitations, this approach must be used in conjunction with manual control.

Chapter 5

Conclusion

This thesis described several tracking techniques and ways of preventing them. Several tools that implement anti-tracking methods have been presented. Afterwards, several ways to compare the anti-tracking tools were described. Based on the existing methods, new comparison methods were presented and their limitations identified.

Future work will include implementing these methods and using them to compare the presented privacy-oriented tools. The comparison results will be visualized and the findings will be commented.

Bibliography

[1] Acar, G.; Eubank, C.; Englehardt, S.; Juarez, M.; Narayanan, A. 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*. New York, NY, USA: Association for Computing Machinery, 2014, p. 674–689. CCS '14. ISBN 9781450329576. Available at: https://doi.org/10.1145/2660267.2660347.

[2] Amazon Web Services, Inc.. *What is Virtualization? - Cloud Computing Virtualization Explained - AWS* online. 2024. Available at: https://aws.amazon.com/what-is/virtualization/. [cit. 2024-10-31].

[3] Barth, A. *HTTP State Management Mechanism* RFC 6265. RFC Editor, april 2011. Available at: https://doi.org/10.17487/RFC6265.

[4] Brave Software. *Tracker Meaning and Definition* online. February 2023. Available at: https://brave.com/glossary/tracker/. [cit. 2024-10-04].

[5] Bujlow, T.; Carela Español, V.; Solé Pareta, J. and Barlet Ros, P. A Survey on Web Tracking: Mechanisms, Implications, and Defenses. *Proceedings of the IEEE*, 2017, vol. 105, no. 8, p. 1476–1510.

[6] Cao, Y.; Li, S. and Wijmans, E. (Cross-)Browser Fingerprinting via OS and Hardware Level Features. In: *Network and Distributed System Security Symposium 2017*. 2017. Available at: https://api.semanticscholar.org/CorpusID:8297353.

[7] Cyphers, B. and Gebhart, G. *Behind the One-Way Mirror: A Deep Dive Into the Technology of Corporate Surveillance* online. Electronic Frontier Foundation, 2019. Available at: https://www.eff.org/wp/behind-the-one-way-mirror. [cit. 2024-10-04].

[8] Datta, A.; Lu, J. and Tschantz, M. C. Evaluating Anti-Fingerprinting Privacy Enhancing Technologies. In: *The World Wide Web Conference*. New York, NY, USA: Association for Computing Machinery, 2019, p. 351–362. WWW '19. ISBN 9781450366748. Available at: https://doi.org/10.1145/3308558.3313703.

[9] Demir, N.; Theis, D.; Urban, T. and Pohlmann, N. *Towards Understanding First-Party Cookie Tracking in the Field*. In:. April 2022.

[10] Eckersley, P. How unique is your web browser? In: *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*. Berlin, Heidelberg: Springer-Verlag, 2010, p. 1–18. PETS'10. ISBN 3642145264.

[11] Empey, Charlotte and Latto, Nica. *VPN Meaning: What Is a VPN & What Does It Do?* online. August 2023. Available at: https://www.avast.com/c-what-is-a-vpn. [cit. 2024-10-09].

[12] Englehardt, S. and Narayanan, A. Online Tracking: A 1-million-site Measurement and Analysis. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2016, p. 1388–1401. CCS '16. ISBN 9781450341394. Available at: https://doi.org/10.1145/2976749.2978313.

[13] Fifield, D. and Egelman, S. Fingerprinting Web Users Through Font Metrics. In: *In Proceedings of the 19th international conference on Financial Cryptography and Data Security*. January 2015, vol. 8975, p. 107–124. ISBN 978-3-662-47853-0.

[14] Frola, J. *AudioContext Browser Fingerprinting*. Brno, CZ, 2022. Bachelor's Thesis. Masaryk University, Faculty of Informatics. Available at: https://is.muni.cz/th/ke5nb/.

[15] Iqbal, U.; Englehardt, S. and Shafiq, Z. *Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors*. 2020. Available at: https://arxiv.org/abs/2008.04480.

[16] Laperdrix, P.; Bielova, N.; Baudry, B. and Avoine, G. Browser Fingerprinting: A Survey. *ACM Trans. Web*. New York, NY, USA: Association for Computing Machinery, april 2020, vol. 14, no. 2. ISSN 1559-1131. Available at: https://doi.org/10.1145/3386040.

[17] Lerner, A.; Simpson, A. K.; Kohno, T. and Roesner, F. Internet Jones and the Raiders of the Lost Trackers: An Archaeological Study of Web Tracking from 1996 to 2016. In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, August 2016. Available at: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lerner.

[18] Mazel, J.; Garnier, R. and Fukuda, K. A comparison of web privacy protection techniques. *Computer Communications*, 2019, vol. 144, p. 162–174. ISSN 0140-3664. Available at: https://www.sciencedirect.com/science/article/pii/S0140366418300604.

[19] Mowery, K. and Shacham, H. Pixel Perfect: Fingerprinting Canvas in HTML5. In: Fredrikson, M., ed. *Proceedings of W2SP 2012*. May 2012.

[20] Munir, S.; Siby, S.; Iqbal, U.; Englehardt, S.; Shafiq, Z. et al. *COOKIEGRAPH: Understanding and Detecting First-Party Tracking Cookies*. 2023. Available at: https://dl.acm.org/doi/10.1145/3576915.3616586.

[21] Petráňová, J. *Porovnání webových rozšíření zaměřených na bezpečnost a soukromí*. 2021. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Available at: https://dspace.vut.cz/items/e620f4fd-3865-4be5-b499-cc38b509a133.

[22] Petráňová, J. *Běhová prostředí pro testování činnosti rozšíření pro webový prohlížeč*. 2024. Master's thesis. Brno University of Technology, Faculty of Information Technology. Available at: https://dspace.vut.cz/items/3fb658b3-70e7-40e5-91d7-1f46f9877a6e.

[23] Polčák, L.; Saloň, M.; Maone, G.; Hranický, R. and McMahon, M. *JShelter: Give Me My Browser Back*. 2023. Available at: https://arxiv.org/abs/2204.01392.

[24] Saloň, M. *Detekce metod zjišťujících otisk prohlížeče*. 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Available at: https://dspace.vut.cz/items/50bb435f-8416-41cf-a3da-db4ac5e2aaba.

[25] Sergey Mostsevenko. *How the Web Audio API is used for audio fingerprinting* online. March 2021. Available at: https://fingerprint.com/blog/audio-fingerprinting/. [cit. 2024-10-10].

[26] Smatana, A. *Evaluation of Little Lies in Browser Fingerprinting*. Brno, CZ, 2024. Bachelor's Thesis. Brno University of Technology, Faculty of Information Technology. Available at: https://www.vut.cz/studenti/zav-prace/detail/152586.

[27] Snyder, P.; Vastel, A. and Livshits, B. Who Filters the Filters: Understanding the Growth, Usefulness and Efficiency of Crowdsourced Ad Blocking. *Proc. ACM Meas. Anal. Comput. Syst.* New York, NY, USA: Association for Computing Machinery, june 2020, vol. 4, no. 2. Available at: https://doi.org/10.1145/3392144.

[28] Starov, O. and Nikiforakis, N. XHOUND: Quantifying the Fingerprintability of Browser Extensions. In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, p. 941–956.

[29] Susnjara, S. and Smalley, I. *What are containers? | IBM* online. May 2024. Available at: https://www.ibm.com/topics/containers. [cit. 2024-10-31].

[30] Takei, N.; Saito, T.; Takasu, K. and Yamada, T. Web Browser Fingerprinting Using Only Cascading Style Sheets. In: *2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*. 2015, p. 57–63.

[31] Traverso, S.; Trevisan, M.; Giannantoni, L.; Mellia, M. and Metwalley, H. Benchmark and comparison of tracker-blockers: Should you trust them? In: *2017 Network Traffic Measurement and Analysis Conference (TMA)*. 2017, p. 1–9.

[32] Zhang, D.; Zhang, J.; Bu, Y.; Chen, B.; Sun, C. et al. A Survey of Browser Fingerprint Research and Application. *Wireless Communications and Mobile Computing*, november 2022, vol. 2022, p. 1–14.