BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

CO-EXISTENCE OF JSHELTER AND FIREFOX RESIST FINGERPRINTING

ROZŠÍŘENÍ JSHELTER V PROHLÍŽEČI FIREFOX S AKTIVNÍ OBRANOU SNÍMÁNÍ OTISKU PROHLÍŽEČE

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **JURAJ FILIP KRAMEC**
AUTOR PRÁCE

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

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

161284

Institut:	Department of Information Systems (DIFS)
Student:	**Kramec Juraj Filip**
Programme:	Information Technology
Title:	**Co-Existence of JShelter and Firefox Resist Fingerprinting**
Category:	Web
Academic year:	2024/25

Assignment:

1. Study the ResistFingerprinting Firefox option, what does it influence and review the current plans for its future improvements.
2. Get familiar with the JShelter project and its protections. Focus on the protection similar to the protections discovered in Point 1.
3. Propose JShelter modifications so that it does not duplicate or apply conflicting modifications in case it is running in Firefox with active ResistFingerprinting. For example, ResistFingerprinting generates random content of 2D canvas and JShelter further modifies the content. Consequently, the modifications by JShelter consume CPU performance but do not improve the fingerprinting protection. Discuss the proposal with the supervisor.
4. Implement the proposal. Share your implementation with the JShelter project for code review.
5. Test the implementation.
6. Discuss your approach, identify its strengths and pitfalls; propose possible future improvements.

Literature:

- Libor Polčák, Marek Saloň, Giorgio Maone a kol. JShelter: Give Me My Browser Back. *Proceedings of the 20th International Conference on Security and Cryptography*. Rome: SciTePress - Science and Technology Publications, 2023, pp. 287-294. ISBN 978-989-758-666-8.
- Jana Petráňová: Běhová prostředí pro testování činnosti rozšíření pro webový prohlížeč. Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií,
- Mozilla Wiki Contributors. Security/Tor Uplift/Tracking. [online] https://wiki.mozilla.org/Security/Tor_Uplift/Tracking, last visit 2024-09-10.

Requirements for the semestral defence:
First three points in a technical report.

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:	14.5.2025
Approval date:	22.10.2024

Abstract

This bachelor's thesis analyses two privacy-enhancing technologies – 'Resist Fingerprinting' (RFP) and 'JShelter'. The two focus on protecting against fingerprinting but employ vastly different mechanisms against it – RFP aims to make all users appear the same, while JShelter randomises fingerprintable values so that its users appear differently on every website, albeit with a high-entropy fingerprint. I first examine these privacy-enhancing tools and their targeted fingerprinting vectors, and then compare their philosophies and interactions. Based on that analysis, I attempt to resolve the overlap with a compatibility mode, which involved heuristic detection of RFP and adjustment of JShelter's protection levels to remove conflict and redundancy on common fingerprinting vectors.

Abstrakt

Táto bakalárska práca analyzuje dve technológie na posilnenie súkromia – ‚Resist Fingerprinting' (RFP) a ‚JShelter'. Obidve sa zameriavajú na ochranu pred snímaním odtlačku prehliadača, ale využívajú proti nemu výrazne odlišné mechanizmy – RFP sa snaží, aby všetci jeho užívatelia vyzerali rovnako, zatiaľ čo JShelter náhodne nastavuje hodnoty odtlačkov tak, aby sa jeho užívatelia na každej webovej stránke zobrazovali inak, hoci s odtlačkom s vysokou entropiou. V tejto práci najskôr skúmam tieto nástroje a ich cielené odtlačkové vektory a potom porovnávam ich návrhovú filozofiu a vzájomné interakcie. Na základe tejto analýzy sa pokúsim vyriešiť ich prekryv režimom kompatibility, ktorý zahŕňal heuristickú detekciu RFP a úpravu úrovní ochrany JShelteru s cieľom odstrániť konflikt a redundanciu na spoločných otlačkových vektoroch.

Keywords

Firefox, JShelter, web browser, browser fingerprinting, web privacy, browser extension, JavaScript

Kľúčové slová

Firefox, JShelter, webový prehliadač, otlačok prehliadača, online súkromie, rozšírenie prehliadača, JavaScript

Reference

KRAMEC, Juraj Filip. *Co-Existence of JShelter and Firefox Resist Fingerprinting.* Brno, Czechia, 2025. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Libor Polčák, Ph.D.

Rozšírený abstrakt

Internet je čím ďalej tým väčšou súčasťou bežného moderného života, a to nielen pre zábavu, ale aj pre prácu, zdravotníctvo, bankovníctvo či osobnú administratívu [11]. Do tohto digitálneho prostredia sa najčastejšie pristupuje pomocou webového prehliadača. Prehliadače sú teda dôležitou súčasťou modernej infraštruktúry a postupom času sa funkcie prehliadačov rozširujú. Okrem zobrazovania stránok HTML môžu komunikovať aj s pripojenými perifériami, ako sú kamery alebo mikrofóny[1], využívať Bluetooth[2] alebo NFC[3], či priamo využívať grafickú kartu na hranie videohier (napr. cez WebGL[4]) alebo využívanie umelej inteligencie (cez WebGPU[5] – napr. Gemma[6] alebo Web Stable Diffusion[7]). Na to všetko musia mať prehliadače volateľné aplikačné rozhrania (API), ktoré odhaľujú mnohé informácie o svojom prostredí. Tieto informácie, získateľné jednoduchým volaním niekoľkých JavaScriptových funkcií, sú zneužiteľné na získanie okamžitého jedinečného „odtlačku“ [9]. Tento odtlačok prehliadača môže byť použitý pre sledovanie správania užívateľov alebo cielený marketing [13]. Čoraz viac užívateľov si uvedomuje, že internet ohrozuje ich súkromie viacerými spôsobmi, a podľa prieskumu z roku 2016, 27 % Európanov používa nejaký nástroj na posilnenie svojho internetového súkromia [6].

V tejto práci sa zameriavam na dva takéto nástroje. Prvým je ‚Resist Fingerprinting‘ (RFP), čo je vstavané nastavenie v prehliadači Mozilla Firefox. Toto nastavenie bolo vytvorené primárne pre derivovaný prehliadač Tor Browser, no aj na obyčajnom prehliadači Firefox, či iných derivátoch, ho možno zapnúť v pokročilých nastaveniach (`about:config`). RFP maskuje mnohé systémové detaily, a namiesto skutočných, rozhrania poskytujú konkrétne štandardizované hodnoty, rovnaké medzi všetkými používateľmi RFP [17, 10]. Niektoré ďalšie rozhrania sú vypnuté alebo inak narušené – konkrétne správanie RFP je popísané v kapitole 2. Cieľom RFP je jednotnosť – nechať užívateľa „splynúť s davom“, aby vypadal rovnako ako všetci ostatní. RFP je štandardom na temnom webe, ale na povrchovom webe sa RFP používa pomerne málo [3].

Druhým zameraným nástrojom na posilnenie súkromia je ‚JShelter‘, čo je rozšírenie prehliadača s viacerými ochrannými modulmi. V tejto práci je hlavným modulom „JavaScript Shield“ (JSS), ktorý obaľuje mnohé funkcie JavaScriptu a mení ich návratové hodnoty. Podľa konfigurácie tohto rozšírenia môžu byť tieto hodnoty mierne poupravené, náhodné alebo skryté (nastavené na nulu, prázdny reťazec atď.) [23, 25]. Konkrétne moduly rozšírenia JShelter, úrovne ochrany modulu JSS a špecifické správanie obalených funkcií v prehliadači Firefox sú opísané v kapitole 3. Predvolená a odporúčaná úroveň ochrany rozšírenia JShelter (tzv. L2) má opačný cieľ ako RFP – jedinečnosť. Rozšírenie nastavuje odtlačkovateľné hodnoty náhodne tak, že prostredie prehliadača vypadá inak na všetkých navštívených doménach.

Cieľom tejto práce bolo nájsť konflikt a prekryv medzi nastavením RFP a rozšírením JShelter, a nejakým spôsobom navrhnúť a implementovať jeho riešenie. V sekcii 4.1 porovnávam rozdiely v prístupe k ochrane súkromia medzi týmito nástrojmi, a následne v sekcii 4.2 konkrétne opatrenia, ktoré sa medzi nimi prekrývajú. Ako je zhrnuté v tabuľke 4.1, k prekryvu dochádza v piatich opatreniach, na ktoré v danej kapitole navrhujem riešenia,

[1] https://developer.mozilla.org/en/docs/Web/API/MediaDevices
[2] https://developer.mozilla.org/en/docs/Web/API/Web_Bluetooth_API
[3] https://developer.mozilla.org/en/docs/Web/API/Web_NFC_API
[4] https://developer.mozilla.org/en/docs/Web/API/WebGL_API
[5] https://developer.mozilla.org/en/docs/Web/API/WebGPU_API
[6] https://ai.google.dev/gemma/docs/integrations/web
[7] https://github.com/mlc-ai/web-stable-diffusion

pričom sa snažím maximalizovať ochranu pred odtlačkovaním, odstrániť zbytočné opatrenia a rešpektovať užívateľské nastavenia. Správanie RFP nie je možné zmeniť, ale keďže JShelter obaľuje volania JavaScriptu, v konečnom dôsledku JShelter určuje návratové hodnoty funkcií.

Na detekciu nastavenia RFP je však potrebné konkrétne povolenie k nastaveniam ochrany súkromia (`privacy`), ktoré JShelter nemá. Z tohto dôvodu neboli navrhované zmeny implementované priamo, ale ako voliteľný „režim kompatibility s RFP“. Keď je tento režim zapnutý, vstavané úrovne ochrany zmenia určité opatrenia podľa tabuľky 4.3. Potrebné povolenie sa od užívateľa vyžiada až po zapnutí tohto režimu. Tlačidlo na jeho povolenie je vložené do nastavení JSS, no aby boli užívatelia oboznámení s jeho existenciou, bola pridaná heuristická detekcia RFP. Táto detekcia sa pokúsi zistiť či užívateľ používa RFP bez priameho prístupu k tomuto nastaveniu, a to testovaním hodnôt, ktoré RFP nastavuje. Pokiaľ je RFP detekované, užívateľovi sa zobrazí notifikácia, informujúca ho o režime kompatibility.

V kapitole 5 opisujem samotnú implementáciu režimu kompatibility s RFP a heuristickú detekciu RFP. Tlačidlo na zapnutie tohto režimu, pridané do nastavení rozšírenia, si po kliknutí najprv vyžiada povolenie `privacy`, a až po jeho udelení nastaví hodnotu `rfpCompatibilityOn`. Pokiaľ je toto nastavenie zapnuté, v rámci súboru `common/levels_cache.js` sa skontroluje, či je RFP zapnuté a či je úroveň ochrany vstavaná. Ak áno, úroveň je pozmenená funkciou `applyRFPAdjustments()`, ktorá je ukázaná v algoritme 5.4. Heuristická detekcia RFP je vložená do spúšťacieho skriptu, aby mohla byť spustená na skutočnej stránke mimo privilegovaného režimu. Testovací algoritmus sa spúšťa len raz za dva dni, a len vtedy, keď nie je nastavené `rfpCompatibilityOn`. Ak algoritmus detekuje RFP s presnosťou nad 66 %, posiela hlásenie, ktorého výsledkom je notifikácia.

Kapitola 6 demonštruje a testuje správnosť a funkčnosť tejto implementácie. Prvý test, opísaný v sekcii 6.1, využíva RFP-Tester[8] – webový detektor RFP – a webovú stránku „Am I Unique?“[9] – webový vyhodnocovač unikátnosti odtlačku prehliadača. Tieto dva nástroje sú spustené so všetkými úrovňami JSS, nastavením RFP a režimom kompatibility – výsledky dokazujú, že režim kompatibility mení stanovené opatrenia JSS a má vplyv na unikátnosť odtlačku prehliadača. Druhý test, opísaný v sekcii 6.2, testuje heuristickú detekciu RFP. Pri vypnutom nastavení RFP testovací algoritmus prebehol na pozadí bez ďalšej akcie, ale pri zapnutom RFP sa zobrazila notifikácia o režime kompatibility. Týmto testom sa zároveň overilo, že testovací algoritmus nie je detekovaný JShelter modulom „Fingerprinter Detector“ (FPD).

Napriek úspešnej implementácii režimu kompatibility s RFP, niekoľko nedostatkov opísaných v sekcii 5.4 zamedzilo vloženiu tejto funkcionality do rozšírenia JShelter. Ide najmä o skutočnosť, že prispôsobenie sa vykonáva v rámci súboru `common/level_cache.js`, čím sa toto prispôsobenie stáva neviditeľným v nastaveniach, a teda aj pre užívateľa. Okrem toho, nemožnosť odhaliť výnimky by nenápadne narúšala niektoré protiopatrenia proti odtlačkovaniu. Keďže však bola implementácia nakoniec úspešná, po vyriešení uvedených problémov, prehodnotení niektorých osobitostí a ďalšom testovaní by táto funkcia mohla byť pre rozšírenie JShelter prínosom pri odstraňovaní redundancie.

[8] https://onegentig.github.io/rfp-tester
[9] https://www.amiunique.org/fingerprint

Co-Existence of JShelter and Firefox Resist Fingerprinting

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Libor Polčák. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Juraj Filip Kramec
14th May 2025

Contents

List of Figures

Chapter 1

Introduction

From social engagement and entertainment to essential services like healthcare or banking, browsers have become the primary gateway to modern life. The Internet serves as a place of work and leisure for many, and even administrative work is becoming more digitised. Last year, the average person spent around 6.4 hours online daily, interacting with web browsers and apps across multiple devices [11, 26]. In this digital environment, web browsers serve as the primary interface through which users access an increasingly essential digital infrastructure. Behind this everyday activity lies a persistent threat: each visit to a website, click, and interaction leaves digital traces behind. All entered information, browsing history, and devices or software in combination create a unique identifier as distinctive as a fingerprint [9].

Obtaining an immediate identifier and reliable of a browser can be as simple as calling a few JavaScript functions. Functionalities of modern browsers go far beyond just displaying simple websites – modern JavaScript functions and interfaces (APIs) allow the development of various web apps, from games through utilities to diverse demos and animations. While well-intended, many of these readily available interfaces expose varied information, such as the used browser environment, hardware specifications, installed fonts or drivers, or general preferences, from which it is possible to create an identifier with high effective entropy. Such an identifier is called a *browser fingerprint*. Although browser fingerprinting may have some constructive uses, e.g. detecting account hijacking, it is often misused for cross-site tracking, profiling or targeted advertising [13].

Many users are aware of such misuse, especially after the global surveillance leaks in the 2010s. A survey by the Pew Research Centre [21] shows that almost 90 % of Americans practice some form of privacy self-help, with 21.9 % employing some *privacy-enhancing technology* (PET). This share rises to 27 % in Europe, according to a Eurobarometer survey [6]. A goal of any PET is to minimise unwanted processing of personal data, preferably without the loss of functionality in the information system [4]. These technologies first identify a vulnerable area of an information system or an interface, which can be used for privacy violation – such areas are called *fingerprinting vectors* – and develop some way to reduce or eliminate the area's vulnerability.

To safeguard against browser fingerprinting, users can utilise various PETs, either as integrated features or browser add-ons. An example of an integrated PET could be

'Resist Fingerprinting'[1] (RFP) – a browser setting in Mozilla Firefox[2], or 'Brave Shields'[3] tracking protection. Notable PET add-ons are multi-platform web extensions 'JShelter'[4] or 'Privacy Badger'[5].

Some users may employ multiple PETs in hopes of stronger privacy protection. When PETs target different fingerprinting vectors, using them together may be beneficial. However, simply using more privacy tools may not necessarily strengthen one's privacy. General PETs addressing various privacy vulnerabilities may have overlapping measures, leading to complex interactions that could negatively impact their performance or effectiveness. Additionally, using them together might paradoxically make the user stand out even more [2].

This thesis takes a look at the interaction between the integrated advanced option of Mozilla Firefox's 'Resist Fingerprinting' and a browser extension JShelter. The goal of this thesis is to analyse the targeted fingerprinting vectors, identify their overlaps, attempt to fix them, and ultimately assess whether their combined application enhances or undermines user privacy.

Chapter 2 examines Firefox's 'Resist Fingerprinting' (RFP) protection, documenting its measures, general philosophy and potential future changes. Chapter 3 then takes a more focused look at JShelter, examining its design differences with RFP and its potentially conflicting protections. The documented measures are then summarised in Chapter 4, comparing individual approaches to each fingerprinting vector and their design philosophy, and eventually suggesting a potential resolution to their conflict. This resolution, including a 'RFP Compatibility Mode' and a heuristic detection of RFP within JShelter, is then implemented in Chapter 5 and tested in Chapter 6.

[1] https://support.mozilla.org/en-US/kb/resist-fingerprinting

[2] 'Resist Fingerprinting' is embedded in the underlying Gecko engine and is present on all browsers derived from Firefox. This option is primarily developed for and used on the Tor Browser, as further explained in Chapter 2.

[3] https://brave.com/privacy-features

[4] https://jshelter.org

[5] https://privacybadger.org

Chapter 2

Firefox Fingerprinting Resistance

At the forefront of advanced Firefox's fingerprinting resistance is the 'Resist Fingerprinting' option built into the browser's JavaScript engine. This option prevents websites from uniquely identifying the user by limiting the information gathered about the user's device. Across various fingerprintable APIs, the browser uses various perturbation techniques (referred to as "spoofing" in Firefox) to mask the system's characteristics. Instead, the browser presents standardised values that match across all users who enable this feature [17, 10]. Attempts at fingerprinting then should, in theory, result in a single identifier, making the fingerprints useless. This behaviour follows the *uniformity approach* to fingerprinting resistance – attempting to create one homogenous fingerprint shared by many users, making it impossible to tell them apart [3].

The Resist Fingerprinting option – RFP for short – was added to Firefox as part of the Tor Uplift project[1] to "land all Tor Browser patches so that Tor can use Firefox's main trunk directly instead of a fork" [19]. On the Firefox browser itself, RFP is turned off by default and is recommended only for advanced users who are comfortable with potential strange behaviours and loss of convenience when browsing [17]. Instead, the 'Enhanced Tracking Protection' is suggested for most users, which blocks cross-site trackers, known fingerprinters and crypto-miners [16]. Few users of Firefox thus use RFP. Of all Firefox traffic intercepted by a fingerprint.com analysis, only 0.48 % matched RFP values [3]. On Tor Browser, however, RFP is enforced and cannot be disabled.

Mozilla Wiki has a list of RFP's perturbations [18]. However, I found the list outdated and incomplete, with some sections missing or empty. Instead of relying on it, I followed Bugzilla tickets and Firefox source code to document and test the measures myself. As a part of this research, I made a web application that attempts to determine whether the user is using RFP. The test suite is not exhaustive, as not all measures can be tested automatically. For example, constructed events do not pass the same perturbation paths as actual events, so this tool does not test measures for `MediaError` messages, Touch API or pointer events. The web application is available at https://onegentig.github.io/rfp-tester.

2.1 Canvas Noise in RFP

Canvas is an HTML5 programmatic drawing surface for displaying dynamic graphics. It exposes a formidable drawing API contained in a `CanvasRenderingContext2D` object [7].

[1] https://bugzilla.mozilla.org/show_bug.cgi?id=1329996

Depending on the renderer, operating system, GPU, installed fonts and font smoothing, or other minor factors, the resulting canvas will vary slightly between devices. This provides a consistent, high-entropy and easily obtainable fingerprint. Fingerprinters can draw a simple image with text and extract the drawn pixels from the application memory. [14]

To prevent canvas-based fingerprinting, RFP adds white noise to extracted canvas data[2]. The noise function applies random noise to randomly selected RGB channels. The pseudorandom number generator is seeded with a context-dependent key, which differs for each domain and session. After this noise is applied, the original image is not recognisable in the canvas. Figure 2.1 shows the randomisation algorithm in pseudocode[3].

Data: canvas*, ctx*

if (canvas.pixels ≤ 4) $\vee$ (ctx $= \varnothing$) $\vee$ (IsUniformColour(canvas)) **then**
  return;
end
key $\leftarrow$ GenerateKey(ctx, canvas); // 32-byte key
$\text{rng}_1 \leftarrow$ InitXorShift128Plus(key[0:8]);
$\text{rng}_2 \leftarrow$ InitXorShift128Plus(key[16:24]);
$n \leftarrow \max \text{key}[31], 20$; // Noise points
while $n > 0$ **do**
  channel $\leftarrow \text{rng}_1$.next() mod 3;
  $i_{\text{pixel}} \leftarrow 4 \cdot (\text{rng}_1.\text{next}() \bmod \text{canvas.pixelCount})$;
  $p \leftarrow i_{\text{pixel}} + \text{channel}$; // Position to modify
  bit $\leftarrow \text{rng}_1$.next() & 1; // Noise bit
  canvas.data[p] $\leftarrow$ canvas.data[p] $\bigoplus (2 \gg \text{bit})$;
  $n \leftarrow n - 1$;
end

Figure 2.1: RFP canvas noise algorithm.

Some web applications render assets on separate off-screen canvases, which are extracted and added to a main visible canvas. In these cases, the user might want to enable canvas extraction. Firefox prompts the user on the first visit, allowing them to add an exception for this domain. This exception can be toggled at any time by the user.

In addition to noise, canvas methods `isPointInPath()`[4] and `isPointInStroke()`[5] always return `false`. These methods are not affected by the per-site exceptions.

2.2 WebGL Properties in RFP

WebGL is a standard 3D graphics library that enables web browsers to render 3D scenes. Based on OpenGL Embedded Systems API, it gives the browser direct access to OpenGL and, through it, the graphics processing unit to display 3D with other HTML content.

[2] https://bugzilla.mozilla.org/show_bug.cgi?id=1816189

[3] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/toolkit/components/resistfingerprinting/nsRFPService.cpp#1695

[4] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/dom/canvas/CanvasRenderingContext2D.cpp#5258

[5] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/dom/canvas/CanvasRenderingContext2D.cpp#5304

WebGL context is accessed through a canvas [8]; hence, the extraction noise described in section 2.1 is likewise applied.

WebGL provides access to low-level drawing methods of the graphics processing unit (GPU) via OpenGL, exposing information about the system's hardware, graphics capabilities and the underlying driver. The WebGL debug renderer info object exposes the GPU manufacturer and model[6].

RFP resolves this fingerprinting vector by implementing a "minimal capability mode", which sets potentially fingerprintable values to a consistent baseline, calibrated to match the lowest common denominator of device capabilities between legacy netbooks and early smartphone models (circa 2013)[7][8]. These values are shown in Figure 2.2. Furthermore, the WebGL debug renderer info extension is disabled[9]. To prevent apps from refusing to render due to weak detected hardware, RFP disables the `failIfMajorPerformanceCaveat` attribute[10].

Property	**Value**
`MAX_TEXTURE_SIZE`	2048
`MAX_CUBE_MAP_TEXTURE_SIZE`	2048
`MAX_RENDERBUFFER_SIZE`	2048
`MAX_VERTEX_TEXTURE_IMAGE_UNITS`	8
`MAX_TEXTURE_IMAGE_UNITS`	8
`MAX_COMBINED_TEXTURE_IMAGE_UNITS`	16
`MAX_VERTEX_ATTRIBS`	16
`MAX_VERTEX_UNIFORM_VECTORS`	256
`MAX_FRAGMENT_UNIFORM_VECTORS`	224
`MAX_VARYING_VECTORS`	8
`MAX_VIEWPORT_DIMS`	4096×4096
`ALIASED_POINT_SIZE_RANGE`	1–63
`ALIASED_LINE_WIDTH_RANGE`	1–1

Figure 2.2: Table of WebGL properties when using minimal capability mode.

The "minimal capability mode" can be enabled separately via the advanced option `webgl.min_capability_mode` advanced option. Tor browser turns off all WebGL extensions and disallows the use of WebGL2[11]. LibreWolf, a privacy-focused fork of Firefox, disables WebGL entirely[12].

[6] https://developer.mozilla.org/en/docs/Web/API/WEBGL_debug_renderer_info

[7] https://gitlab.torproject.org/tpo/applications/tor-browser/-/commit/1acd0c7fae9121240401cf4a8f0e2b1f6fdb9827

[8] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/dom/canvas/test/webgl-mochitest/test_webgl_fingerprinting_resistance.html

[9] https://bugzilla.mozilla.org/show_bug.cgi?id=1337157

[10] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/dom/canvas/ClientWebGLContext.cpp#926

[11] https://2019.www.torproject.org/projects/torbrowser/design#fingerprinting-linkability

[12] https://librewolf.net/docs/features#privacy

2.3 Time Precision Reduction in RFP

Computers have many internal clocks for different purposes, often consisting of multiple hardware and software components. From different internal ticks and manufacturing variations to small synchronisation drifts, even the temperature of the clock crystal can slightly affect the system time. While these variations are imperceivable to the user, they can reveal information about the state of a system. These variations are often calculated or estimated as *clock skew* – the rate of change of offsets. Clock skew tends to stay stable on one machine ($\pm$ 1–2 ppm) while varying between machines (up to $\pm$ 50 ppm) enough to form a viable and remotely measurable fingerprinting metric [20, 24].

In addition to privacy concerns, precise timing measurements are also a security concern. An example of a time-based attack could be Spectre [12]. By measuring tiny timing differences in memory access speeds, an attacker can infer when data is and is not in a cache and use this knowledge to leak sensitive information. Following the disclosure of Spectre in 2018, browser developers became more proactive in mitigating timing attacks [24]. Since Firefox 57 (released in November 2017), all timestamps are rounded to 20 microseconds[13].

RFP uses the exact rounding method with an increased resolution of 16.667 milliseconds (corresponds to a 60 Hz refresh rate) and temporal jitter[14]. Instead of rounding to the nearest resolution multiple, a pseudorandom midpoint is generated for every rounding[15].

2.4 Media Devices Perturbation in RFP

The Media Devices API is an interface to connected audiovisual peripherals, such as speakers, cameras or microphones. By calling `navigator.mediaDevices.enumerateDevices()`, a web application can obtain a complete list of such peripherals, labels, and unique identifiers[16]. This forms a viable hardware-based fingerprinting vector.

Firefox ties the device identifiers with the browsing context even when RFP is turned off. Each website will see a different identifier for the same device, and clearing the browser's cache for its domain will cause the identifier to be regenerated[17]. With RFP turned on, Firefox will mask the peripherals entirely. Only one peripheral of each kind (audio input, audio output, and video input) is kept and renamed to a generic name for its type – 'Internal Microphone', 'Internal Speakers' and 'Internal Camera', respectively. Other peripherals are hidden and do not appear in the array. If there are no peripherals of a certain kind (e.g. no camera is connected), a dummy device is added, appearing as an actual device. The order of devices is always microphone → camera → speaker[18].

[13] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/toolkit/components/resistfingerprinting/nsRFPService.cpp#716

[14] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/toolkit/components/resistfingerprinting/nsRFPService.cpp#520

[15] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/toolkit/components/resistfingerprinting/nsRFPService.cpp#544

[16] https://developer.mozilla.org/en/docs/Web/API/MediaDevices

[17] https://developer.mozilla.org/en/docs/Web/API/MediaDeviceInfo/deviceId

[18] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/dom/media/MediaDevices.cpp#353

2.5 Other Perturbations in RFP

- Timezone is set to Atlantic/Reykjavik (year-round UTC)[19].
- Vendor (`navigator.vendor` and `navigator.vendorSub`) is not defined (empty string).
- Several `navigator` properties are set according to the client's operating system; see Figure 2.4.
- The build ID is always 20181001000000[20].
- The number of logical processors (`navigator.hardwareConcurrency`) is always 2[21].
- Pointer events have a uniform type and a 0° angle[22].
- The framerate of WebRenderer is locked at 60 FPS[23].
- Size-specific zoom is disabled[24].
- Audio output latency is set according to the client's operating system[25]; see Figure 2.3.
- Media errors cannot be set to arbitrary text[26].
- Screen orientation is set to "landscape-primary" (width ≥ height) or "portrait-primary" (otherwise)[27].
- Colour depth (`screen.colorDepth`) and pixel depth (`screen.pixelDepth`) are always 24-bit.
- One CSS pixel corresponds to two physical pixels (`window.devicePixelRatio`).
- Various values in Screen API are spoofed[28].
- sRGB colour space (gamut) is enforced.
- Various CSS media preferences, such as reduced motion or dark mode, are disabled[29].

Windows	**Mac**	**Android**	**Linux**
0.04	512/`mSampleRate`	0.02	0.025

Figure 2.3: Table of OS-dependant audio latency set by RFP.

[19] https://bugzilla.mozilla.org/show_bug.cgi?id=1330890
[20] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/browser/components/resistfingerprinting/test/browser/browser_navigator.js
[21] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/dom/workers/RuntimeService.cpp#1930
[22] https://bugzilla.mozilla.org/show_bug.cgi?id=1363508
[23] https://bugzilla.mozilla.org/show_bug.cgi?id=1772711#c1
[24] https://bugzilla.mozilla.org/show_bug.cgi?id=1369357
[25] https://hg-edge.mozilla.org/integration/autoland/rev/c96e81ba64f3
[26] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/dom/html/MediaError.cpp#34
[27] https://bugzilla.mozilla.org/show_bug.cgi?id=1281949
[28] https://searchfox.org/mozilla-central/source/dom/base/test/chrome/bug418986-1.js
[29] https://bugzilla.mozilla.org/show_bug.cgi?id=1479240

Property	Windows	Mac	Android	Linux
appVersion	5.0 (Windows)	5.0 (Macintosh)	5.0 (Android 10)	5.0 (X11)
platform	Win32	MacIntel	Linux armv81	Linux x86_64
oscpu	Windows NT 10.0; Win64; x64	Intel Mac OS X 10.15	Linux armv81	Linux x86_64

Figure 2.4: Table of OS-dependant RFP values in the `navigator` interface.

2.6 APIs Disabled by RFP

- Performance API, which would expose granular timing data about DNS lookups and resource loading times, has its values modified to report immediate loading[30].
- Web Speech API, which would expose the client's speech synthesis back-end and installed voices, is disabled. The voices array is always empty, and calling the `speak()` function throws an error[31].
- Gamepad API, which would expose connected game controllers, always returns an empty array[32].

2.7 Potential Future Changes to RFP

- Sensors API was proposed in 2018 but remains unimplemented[33]. If added to Firefox, it would form a significant fingerprinting vector[34].
- Web Speech API currently only implements speech synthesis but has no speech-to-text capability. Depending on the implementation, it could form a fingerprinting vector.
- The current timezone perturbation, listed in Section 2.5, is criticised for significantly hindering usability and confusing users. There are discussions about changing its approach, for example, making it a separate setting or mapping less common timezones to their more prevalent equivalents (for example, mapping all timezones following CET/CEST to Europe/Berlin)[35].
- Mozilla is slowly rolling out a more fine-grained alternative to RFP that allows granular control over used perturbations, e.g. using RFP only in private browsing mode or turning off certain perturbations for specific domains. These options are collectively called "Fingerprint Protection" – FPP for short[36].

[30] https://bugzilla.mozilla.org/show_bug.cgi?id=1369303
[31] https://bugzilla.mozilla.org/show_bug.cgi?id=1333641
[32] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/dom/base/nsGlobalWindowInner.cpp#6766
[33] https://bugzilla.mozilla.org/show_bug.cgi?id=1432631
[34] https://bugzilla.mozilla.org/show_bug.cgi?id=1425130
[35] https://bugzilla.mozilla.org/show_bug.cgi?id=1364261
[36] https://bugzilla.mozilla.org/show_bug.cgi?id=1450398

Chapter 3

JShelter on Firefox

JShelter is an open-source web browser extension that mitigates potential privacy and security threats exploitable with client-side JavaScript. Working on a higher level, JShelter implements a comprehensive layer of wrappers around JavaScript calls, enabling fine-grained control over how these interfaces behave and interact with web applications [23, 25]. JShelter comprises three key protection modules:

1. *JavaScript Shield* (JSS) modifies browser APIs' behaviour to hide and fake fingerprintable information and protect against JavaScript-based attacks. As of 2024, JSS consists of 35 wrappers. JSS offers four protection levels:
   - L0: JSS is turned off.
   - L1: JSS only applies security measures with no fingerprinting resistance.
   - L2: JSS makes the browser appear differently across domains by randomising fingerprintable values. This is the default and recommended level of protection.
   - L3: JSS floors or removes fingerprintable values with the goal of heavy obfuscation of the client system.
2. *Network Boundary Shield* (NBS) prevents browser misuse as a local and public network proxy.
3. *Fingerprint Detector* (FPD) monitors APIs commonly misused for fingerprinting, notifying users when too many sensitive calls are detected. If configured to do so, FPD can block further calls or HTTP requests to the suspected fingerprinter.

Unlike Firefox's 'Resist Fingerprinting' (RFP), JShelter follows the *uniqueness approach* to fingerprinting resistance by default – it aims to create a unique fingerprint for each domain and session, making it impossible to link users across domains [23]. The two privacy-enhancing tools thus employ fundamentally different strategies targeting common fingerprinting vectors, possibly resulting in contradictions. On L3, however, JShelter follows the uniformness approach like RFP. The overlap here should not be contradictory but may be redundant at the cost of performance.

RFP and JShelter may conflict with the two privacy-enhancing tools enabled in any Firefox-based browser. IceCat[1], a Firefox fork by GNU, comes with both tools enabled out of the box.

[1] https://www.gnu.org/software/gnuzilla

This chapter aims to find measures of JShelter JSS that target the same fingerprinting vectors as RFP. Chapter 4 will then focus on their interaction and propose resolutions to their conflict.

3.1 Canvas Modifications in JShelter

JShelter implements two approaches to preventing canvas-based fingerprinting. The first approach (clearing) makes any extracted canvas appear as a plain white image[2]. Methods `isPointInPath()`[3] and `isPointInStroke()`[4], as if the canvas was blank, always return a false Boolean value. The second approach (perturbation, also referred to as "farbling" in JShelter and Brave browser) is more subtle, routing the canvas through a randomisation function that changes the canvas before extraction. Figure 3.1 shows the randomisation in pseudocode[5]. Methods `isPointInPath()`[3] and `isPointInStroke()`[4], while using the perturbation approach, have a 5 % chance to return a flipped Boolean value and a 95 % chance to return the actual value.

Data: canvas*, domainHash

crc ← CRC16();
foreach canvas.row **do**
  crc.next(canvas.row);
end
rng ← alea(domainHash, "CanvasFarbling", crc.crc());
foreach p *in* canvas.data **do**
  foreach channel *in* p **do**
    if $\text{channel}_i = 4$ **then**
      continue; // Skip alpha channel
    end
    if rng.bit() **then** // 50 % chance
      $p = p \bigoplus 1$; // Flip last bit (± 1)
    end
  end
end

Figure 3.1: JShelter canvas noise algorithm.

3.2 WebGL Properties in JShelter

To prevent `WebGLRenderingContext` from exposing hardware information and graphics capabilities, JShelter modifies the return values of API calls, internal methods, and image

[2] https://pagure.io/JShelter/webextension/blob/aaefb7105c048daa09de20b24edf900c145fbc7d/f/common/wrappingS-H-C.js#_139
[3] https://pagure.io/JShelter/webextension/blob/aaefb7105c048daa09de20b24edf900c145fbc7d/f/common/wrappingS-H-C.js#_269
[4] https://pagure.io/JShelter/webextension/blob/aaefb7105c048daa09de20b24edf900c145fbc7d/f/common/wrappingS-H-C.js#_306
[5] https://pagure.io/JShelter/webextension/blob/aaefb7105c048daa09de20b24edf900c145fbc7d/f/common/wrappingL-CANVAS.js

data extraction. Method `readPixels()` routes the output through the same randomisation function as Figure 3.1. For many properties and return values, JShelter employs two possible approaches, similar to its canvas measures described in Section 3.1.

The first approach (perturbation) will multiply integer properties by a random number and set all string properties, such as the vendor or renderer, to random eight-character long strings. The second approach (flooring) returns zeroed values – string properties return empty strings, numbers are set to the lowest possible, and image extraction returns an empty image[6]. Both these approaches are based on the Brave browser[7].

3.3 Time Precision Reduction in JShelter

JShelter implements a "Performance Timeline" wrapper to prevent time-related attacks and fingerprinting[8], as described in Section 2.3. The wrapper can be configured to round all timestamps with precision p or add noise. The rounding function follows the equation in Figure 3.2. The noise function follows a monotonic noise generator that separately applies modulo-based noise to integer and decimal parts of a number. The function maintains a state to ensure rising monotonicity. Figure 3.3 shows the noise generator in pseudocode[9].

$$\dot{x} = x - x \bmod 10^{\max 3-p,0}$$

Figure 3.2: JShelter rounding equation.

Input: n (number to add noise to), p (precision)
Output: x (number with noise)
Data: x_{last} (previous value)

noise $\leftarrow \lfloor$`Random()` $\cdot\, 10^{3-p} \rfloor$;
$n_{\text{whole}} \leftarrow \lfloor n \rfloor$; `// Separate whole`
$n_{\text{decimal}} \leftarrow n - n_{\text{whole}}$; `// and decimal parts`
$x_{\text{whole}} \leftarrow n_{\text{whole}} - (n_{\text{whole}} \bmod$ noise$)$;
$x_{\text{decimal}} \leftarrow n_{\text{decimal}} - (n_{\text{decimal}} \bmod$ noise$)$;
$x \leftarrow x_{\text{whole}} + x_{\text{decimal}}$;
if $x < x_{last}$ **then**
  $x \leftarrow x_{\text{last}}$; `// Ensure monotonicity`
else
  $x_{\text{last}} \leftarrow x$;
end
return x;

Figure 3.3: JShelter numerical noise algorithm.

[6] https://pagure.io/JShelter/webextension/blob/aaefb7105c048daa09de20b24edf900c145fbc7d/f/common/wrappingS-WEBGL.js
[7] https://jshelter.org/webgl
[8] https://jshelter.org/pt2
[9] https://github.com/polcak/jsrestrictor/blob/2fb4cf27ee304d01f4f1f586e092bd9d52b2b190/common/wrapping.js#L59

3.4 Media Devices Spoofing in JShelter

To prevent fingerprinting of the connected media devices, as explained in Section 2.4, JShelter wraps the `MediaDevices.enumerateDevices()` method and modifies the returned array of devices in one of three ways. The first approach is to shuffle the array without modifying or adding any devices. The random number generator used for the shuffling uses the domain hash as a part of the seed, tying the order to the service. The second approach also shuffles the array but adds in a few fake devices generated following a domain-seeded generator. The number of faked devices follows the $Uniform(0, 4)$ distribution. The third approach does not attempt to create a unique fingerprint, instead returning an empty array[10]. This way, no media devices are exposed, though it may lead to breaking services relying on audiovisual input and output.

3.5 Other APIs Disabled by JShelter

JShelter is a cross-platform project; hence, it also affects APIs that are not implemented in Firefox, such as `IdleDetector` (only on Chromium[11]) or `NDEFReader` (only on Android Chromium[12]). The following list only focuses on JShelter's Firefox protections and is not exhaustive. The complete list of JShelter's protections and wrappers can be found on the project's official website [1].

- Array buffers can either be shifted by a fixed offset or have their items randomly remapped[13].
- Background Tasks API returns fake information about remaining time, perturbing cooperative scheduling[14].
- The list of supported codecs is either modified or all codecs are shown as unsupported[15].
- The `MediaKeySystemAccess` object, which allows querying supported encryption mechanisms of the system, is either perturbed (a non-existing key system is added) or disabled entirely.[16].
- `SharedArrayBuffer` is modified to prevent obtaining dangerously precise timestamps (with a mechanism similar to one described in Section 3.3).[17].
- The Geolocation API has reduced precision, and the reported location remains stationary[18].
- `XMLHttpRequests` (XHR) data transfers may be blocked, depending on the user's configuration[19].

[10] https://jshelter.org/mcs
[11] https://developer.mozilla.org/en/docs/Web/API/IdleDetector
[12] https://developer.mozilla.org/en/docs/Web/API/NDEFReader
[13] https://jshelter.org/ecma-array
[14] https://jshelter.org/coop-scheduling
[15] https://jshelter.org/html5
[16] https://jshelter.org/eme
[17] https://jshelter.org/ecma-shared
[18] https://jshelter.org/geo
[19] https://jshelter.org/ajax

Chapter 4

Design Decisions

Both 'Resist Fingerprinting' (RFP) and 'JShelter' are privacy-enhancing technologies with a wide range of targeted fingerprinting vectors. While the previous chapters explored them in separation, this chapter discusses the two tools together, discussing their overlap, interactions and differences in overall approach to fingerprint resistance.

4.1 Philosophy

RFP and JShelter are privacy-enhancing technologies with fundamentally different approaches to fingerprinting resistance. RFP follows the uniformness approach ("blend into the crowd"), while JShelter follows the uniqueness approach ("appear different everywhere"). Both these tools target a wide range of fingerprint vectors; hence, regardless of the number of overlapping targeted vectors, the impact on the fingerprint is bound to be noticeable. These different approaches stem from their intended use cases and technical constraints.

'Resist Fingerprinting' is focused on establishing an anonymity set – a set of users with all the same attributes, hence the same fingerprint, making them indistinguishable [22]. The idea is that when a fingerprinter obtains an RFP fingerprint, the information gained from it is minimal, as many users share that exact fingerprint. It aims to minimise the amount of distinguishing data (known as Shannon information or surprisals in information theory). This approach – using standardised values – is what W3C recommends as the more effective approach to fingerprint reduction [27]. However, an anonymity set is only as good as its size. Standardising fingerprintable values works best in controlled environments, where an anonymity set can be enforced, e.g., Tor Browser has RFP forced on with further masking, such as the use of onion routing. On the clear web, however, only 0.48 % of all Firefox traffic matched values set by RFP [3], diminishing the value of the set. In an uncontrolled space like the clear web, the already small anonymity set further is prone to being fractured by different configurations (overrides), window sizes, etc.

'JShelter' operates within the confines of a web extension made for the clear web. The various wrappers implemented by the 'JavaScript Shield' module focus on randomising fingerprintable values to prevent cross-site tracking (on the 'Recommended' L2 level) or clearing/zeroing the values to prevent sharing information about its environment (on the 'Strict' L3 level). In terms of anonymity, especially on L2, it does not establish any anonymity set – the randomised values are high in entropy. The goal of JShelter is to make sure the user has different values on different domains. A simple fingerprinter would not be able to correlate

the user's presence across domains despite the high conditional entropy. While the W3C fingerprinting guidance document argues that randomisation efficacy is hard to measure and less effective in fingerprint resistance [27], the JShelter design document argues that this is a way of embracing the clear net, where establishing an anonymity set is too difficult [23].

The use of both privacy-enhancing tools creates a protection system with overlapping yet also complementary elements. While the use of JShelter along with RFP reduces the anonymity set (theoretical concern), it can provide better protection against cross-domain tracking than RFP alone (practical benefit). From JShelter's standpoint, RFP can more effectively mask specific fingerprinting vectors (e.g. certain events, TTS voices, fonts) as it operates at a lower level.

4.2 Interactions

When both protective measures are active simultaneously, their interactions vary based on the protected fingerprinting vector. It is important to note that RFP's perturbations will be applied regardless of the proposal, as it operates on a lower level. JShelter's wrappers receive an already-perturbed output of targeted JavaScript functions. However, ultimately, JShelter decides the final returned values.

The resolution should respect the user's chosen protection level, maximise fingerprinting protection, remove redundancy and minimise computational overhead.

Target	Resist Fingerprinting	JShelter	
Canvas API	Noise (Heavy)	Noise (Light)	Clear
WebGL properties	Hard-coded	Random	Clear
Time precision	Round + Jitter	Round	Noise
Media devices	Hard-coded	Shuffle + Random	Clear
Gamepads	Clear		

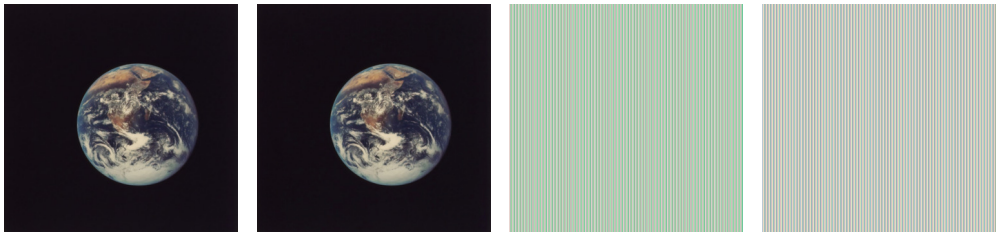
Figure 4.1: Summary table of overlapping perturbations.

4.2.1 Canvas

RFP and JShelter both add noise to canvas extracts using a pseudorandom number generator seeded with the browsing context. Their noise algorithms, shown in Figures 2.1 and 3.1, vary significantly in intensity: Figure 4.2 shows how a re-extracted image[1] looks with the added noise. JShelter-added noise is indistinguishable to the human eye, only modifying each colour channel by one unit. In contrast, RFP-added noise heavily distorts the image down to white noise.

Proposed modifications: As RFP noise function uses a context-based seed, JShelter noise does not yield further privacy benefits. It sufficiently protects against fingerprinting and exposing system information. I thus propose entirely turning off JShelter canvas perturbation on all protection levels.

[1] Image by NASA, 1972, published by New York Public Library: https://unsplash.com/photos/planet-earth-close-up-photography-yEauzeZU6xo



(a) No noise
0571b2d53a2...fb76342098

(b) JShelter
6da8635912d...64eb010752

(c) RFP
6403657018c...7eba088669

(d) RFP + JShelter
558a981a458...e396c0fb1e

Figure 4.2: Comparison of canvas extracts and their SHA256 hashes.

4.2.2 WebGL Properties

RFP sets WebGL properties to standardised values, as shown in Figure 2.2, which JShelter subsequently randomises. This interaction effectively results in JShelter's perturbation prevailing. As RFP values are hard-coded, the interaction does not cause performance overhead. On L3, JShelter sets WebGL properties to zero. While it succeeds at obfuscating client system information, it renders WebGL unusable and is itself fingerprintable across domains. The "minimal capability mode", described in Chapter 2.2, would yield the same privacy benefit while allowing WebGL to function and the fingerprint to blend in with the RFP anonymity set.

Proposed modifications: JShelter randomisation should be kept as-is on L2. On L3, the "minimal capability mode" is more advantageous to privacy and usability. JShelter clearing should be turned off.

4.2.3 Time Precision Reduction

Time-based fingerprinting protection in both tools creates a complex interaction. RFP implements a consistent jitter at a 60 Hz interval, rounding the timestamp to "bins". This jitter can cause a timestamp to round to a lower bin, practically making time go backwards. JShelter applies more varied random noise with upward monotonicity. While this combination theoretically enhances uniquity by adding further randomisation to the jitter, it introduces significant performance overhead.

Proposed modifications: As RFP adds rounding and pseudorandom noise to timestamps, JShelter rounding and noise are redundant and can be turned off.

4.2.4 Media Devices

RFP enforces device obfuscation and consistent order, with the `MediaDeviceInfo` enumeration of devices always appearing the same. JShelter, by default, adds fake devices and shuffles the array. The performance overhead of shuffling a three-member array is negligible. On L3, JShelter removes all devices from the enumeration. Similar to WebGL properties, described in Section 4.2.2, while succeeding at its goal, it makes media devices unusable. RFP can obfuscate connected devices while retaining their functionality.

Proposed modifications: On L2, JShelter perturbation is more effective without significant performance overhead and should be kept as-is. On L3, utilising RFP enforced order and fake devices obfuscates client system information with further benefits. JShelter clearing should thus be turned off.

4.2.5 Gamepads

The Gamepad API represents the simplest case of an overlap. RFP and JShelter cause the enumeration of gamepads to be empty at all times, making the API unusable. The removal is redundant but has no performance impact. Unlike JShelter, RFP prevents `gamepadconnected` and `gamepaddisconnected` events.

The Gamepad API represents the simplest case of an overlap. RFP and JShelter both make the API unusable by returning an empty array, reporting no controllers. Furthermore, RFP prevents execution of `gamepadconnected` and `gamepaddisconnected` events, which JShelter does not.

Proposed modifications: JShelter perturbation can be removed, as RFP already clears the gamepad array.

4.3 Generalisation

So far, the proposed modifications in section 4.2 have been limited to JShelter's built-in levels, L2 and L3. The proposal can be generalised to a conditional mapping between measure modes to include all built-in and potentially user-defined levels. Rather than hard-coding adjustments per level, compatibility with RFP can be implemented as a set of conditional rules. This would make the adjustment more flexible and easier to manage. The conditional rules implied from the proposed modifications are listed in Table 4.3.

Measure	Original mode	Adjusted mode
Time precision	x (Any)	0 (Disabled)
Gamepad API	x (Any)	0 (Disabled)
Canvas	1 (Noise)	0 (Disabled)
WebGL properties	2 (Floor)	0 (Disabled)
Media devices	3 (Remove)	0 (Disabled)

Figure 4.3: Table of conditional adjustment rules for proposed RFP compatibility mode.

These adjustments can also be made to user-defined levels; however, unlike built-in levels, user-defined levels may have specific needs and intentions that these adjustments would not address. Therefore, I decided to apply these adjustments only to the built-in levels.

4.4 Detection

Extensions can reliably determine whether 'Resist Fingerprinting' is enabled via the `privacy.websites.resistFingerprinting`[2] browser setting, which requires the 'privacy' permission. However, JShelter currently lacks this permission, and adding it could cause issues. According to Libor Polčák, adding a further required permission to an existing extension creates adoption barriers, as many users miss or ignore permission grant prompts on updates, effectively remaining on older versions (personal communication, 15th January 2025).

[2] https://developer.mozilla.org/en/docs/Mozilla/Add-ons/WebExtensions/API/privacy/websites#resistfingerprinting

Therefore, 'privacy' will be included as an optional permission, which will only be requested when a user activates the RFP compatibility mode.

This can lead to a situation where a user utilises RFP and JShelter together without knowing about the compatibility mode. A heuristic method for detecting RFP was thus suggested to inform users about the setting. The detection will utilise research from Chapter 2 and incorporate code from the RFP-Tester[3] test suite developed during that research.

[3] https://onegentig.github.io/rfp-tester, source code available at https://github.com/onegentig/rfp-tester

Chapter 5

Implementation

This chapter describes the practical implementation of functionalities outlined in Chapter 4. Section 5.1 first introduces the new opt-in `rfpCompatibilityOn` flag and its setting in JShelter's options. Section 5.2 then focuses on implementing a function that adjusts the current level to Resist Fingerprinting (RFP) and how it is invoked. Section 5.3 then tackles the heuristic detection of RFP, which is intended to inform the user of the compatibility mode's existence. Lastly, Section 5.4 discusses the limitations and issues of the adjustment and detection functions and what other features were considered and dismissed (and why).

5.1 Compatibility Setting

Following the proposed changes in Section 4.2, generalised in Section 4.3, the built-in levels of JShelter will be adjusted when RFP is detected. This adjustment relies on precise RFP detection, which cannot be done without the 'privacy' permission that JShelter currently lacks. As such, the compatibility mode was implemented as an opt-in setting, which asks the user for this permission when toggled. To enable this, the permission was added to the `optional_permissions` array in the Firefox-specific `manifest.json` file.

This compatibility mode setting is implemented as a new boolean flag stored in the extension's synchronised storage – `rfpCompatibilityOn`. This follows the extension's standing naming and storing conventions.

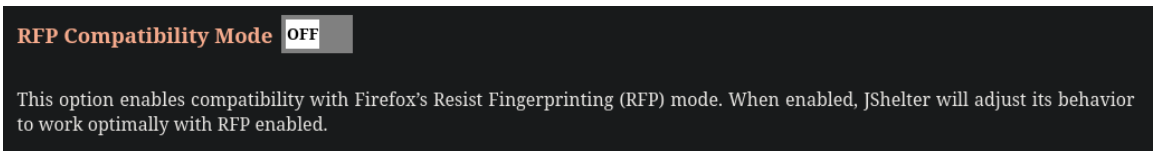


Figure 5.1: RFP compatibility mode toggle in JShelter options.

This setting is managed by a new toggle added to the `common/options.html` file; its section was added under JavaScript Shield settings. The functional logic is handled in the `common/options.js` file. On page load (when the `DOMContentLoaded` event is intercepted), it is first checked that the extension is running on Firefox by analysing the user agent string [15]. While this is a discouraged practice, the preferred check – feature detection – is impossible without having 'privacy' permission, and I wanted to avoid checking for arbitrary differences between browsers that may change in the future.

If it is determined that the environment is not Firefox, the RFP compatibility section will be removed from the DOM; otherwise, it will add a "change" event listener to the toggle. After that, the `rfpCompatibilityOn` setting is checked, and the toggle is set accordingly. Turning the setting off removes the flag from the extension's synchronised storage. Turning it on will first check for the 'privacy' permission. The user will be prompted to allow the extension to read and modify their privacy settings if not given. If the request is denied, the toggle switches back to 'off'; if granted, it displays 'on', and the `rfpCompatibilityOn` flag is stored.

JShelter provides an easy way to translate the text on its pages through the `common/i18n_translate_dom.js` file. Adding the title and description strings to `common/_locales/{en,cs,ru}/messages.json` shows the text in English (default; shown in Figure 5.1), Czech (shown in Figure 5.2) and Russian (shown in Figure 5.3), based on the browser's language settings.

Figure 5.2: RFP compatibility mode toggle in Czech language.

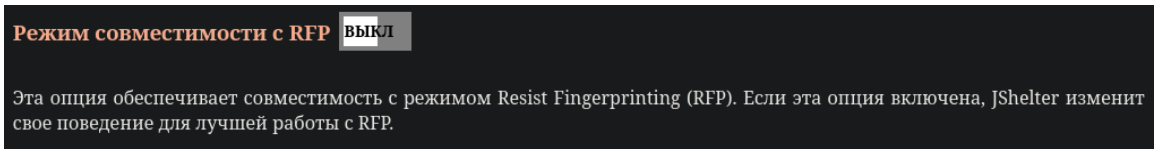


Figure 5.3: RFP compatibility mode toggle in Russian language (translated by Zoltán Kramec).

5.2 Level Adjustment

The core functionality of the RFP compatibility mode is adjusting JShelter's protection levels when RFP is detected per the generalised proposal shown in Table 4.3. For this purpose, I created a new function in `common/levels.js` called `applyRFPAdjustments()` that takes a protection level configuration as an input and returns a modified configuration that avoids conflicts with RFP. The mode mapping from Table 4.3 is stored as a key-value constant object within the function, replacing the defined original modes. This function is shown in Figure 5.4.

This function is invoked in the `common/levels_cache.js` file during configuration loading, replacing the current level. It is called only if all these three conditions are met:

1. The `rfpCompatibilityMode` flag is set,
2. The level is built-in (as decided in Section 4.3),
3. RFP is enabled.

```
async function applyRFPAdjustments(level) {
    if (!levels_initialised) return level;
    const adjustments = {
        time_precision: 0,
        gamepads: 0,
        webgl: (level.webgl === 2) ? 0 : level.webgl,
        htmlcanvaselement:
            (level.htmlcanvaselement === 1)
                ? 0 : level.htmlcanvaselement,
        enumerateDevices:
            (level.enumerateDevices === 3)
                ? 0 : level.enumerateDevices,
        };

        for (const [feat, adj] of Object.entries(adjustments)) {
                if (level[feat] !== undefined) {
                        level[feat] = adj;
                }
        }

        return level;
}
```

Figure 5.4: Adjustment function of levels for RFP compatibility.

To check if RFP is enabled, another function had to be implemented in `common/levels.js` – `isRFPEnabled()` – which checks that it is being executed in a Firefox environment, verifies that the necessary permissions are granted, and includes appropriate error handling before assessing RFP itself. If the function detects that `rfpCompatibilityOn` is enabled but the 'privacy' permission is missing or if the browser is not Firefox, it will turn this setting off. A flowchart of this function is shown in Figure 5.5.

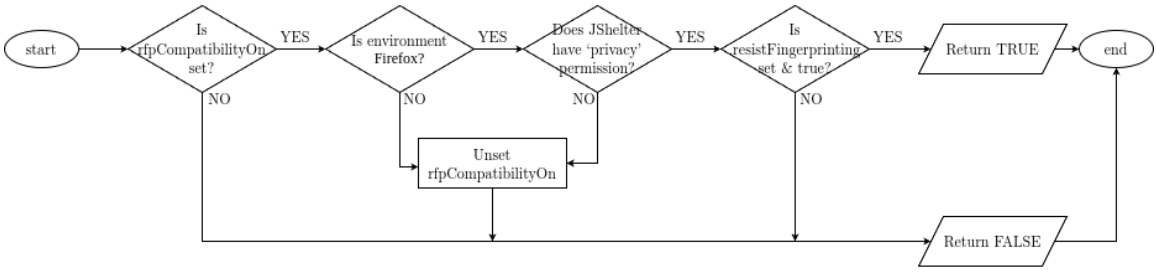


Figure 5.5: Flowchart of the RFP-checking function.

The RFP checking and level adjustment itself happen in the `common/level_cache.js` file within the `getContentConfiguration()` function. After the level is obtained and serialised to JSON, the aforementioned expression evaluates whether it should be adjusted for RFP presence. If so, the `applyRFPAdjustments()` function directly modifies the protection level.

5.3 Heuristic Detection

Various factors presented challenges in implementing a heuristic detection method for RFP. The original idea was to make a background script[1] to detect RFP without bothering or even notifying the user. Unfortunately, this proved impossible because all background scripts and extension pages operate within a so-called "privileged context"[2]. The method `nsContentUtils::ShouldResistFingerprinting()`[3] in Firefox checks for many exceptions and other settings, including exempted domains and schemas. Logical placements for the test suite, such as the background scripts, would run with the `moz-extension://` schema, which is exempted from RFP entirely, along with Firefox internal pages and resources[4]. As such, the test suite must be injected into the context of the actual website.

The file `common/document_start.js` was a logical place for an on-start script. This file, specifically its `configureInjection()` function, handles loading the page configuration (such as exempted or disabled protections) and ends with the initialisation of Fingerprint Detector (FPD) and JavaScript Shield (JSS) wrappers. This function is executed just after the `getContentConfiguration()` function, which, among other things, adjusts the level for RFP. This means that user-set exemptions would still be applied regardless of RFP compatibility changes (which is good in case any changes introduce site breakage). However, an unfortunate factor of the `configureInjection()` function is that this function is synchronous. Hence, the test suite and all its assertions must also be synchronous. Fortunately, most of the tests from the RFP-Tester are synchronous, making only a small set of tests ineligible for this injected test suite.

```
1 const rfpTests = [
2         {
3                 name: "Timezone & Locale",
4                 weight: ACCURACY.MEDIUM + RELIABILITY.MEDIUM,
5                 difficulty: DIFFICULTY.LOW,
6                 execute: checkLocale,
7         },
8         /* ... other tests ... */
9 ];
```

Figure 5.6: Example of a test definition in `firefox/rfp_tests.js`.

A new file was created in the `firefox` folder for the tests and the testing function itself, as it is intended only to be included in the Firefox version. The tests are defined in an object array `rfpTests`, each with a unique name, weight (how accurately can it detect RFP) and reliability (that its passing is not affected by other extensions or natural matters), difficulty (amount of hardware resources required to run the test) and an execution function

[1] https://developer.mozilla.org/en/docs/Mozilla/Add-ons/WebExtensions/Background_scripts
[2] https://developer.mozilla.org/en/docs/Glossary/Privileged_code
[3] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/dom/base/nsContentUtils.cpp#2559
[4] https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/dom/base/nsContentUtils.cpp#2450

(that returns a Boolean value[5] and is strictly synchronous). Figure 5.6 shows the format of this array with the first implemented test, which tests for locale and time zone.

The primary function of the `firefox/rfp_tests.js` file is `isRFPLikely()`, which runs all tests and calculates the likelihood of RFP being on. An algorithm of this test runner is shown in Figure 5.7. This function is globally exposed as `window.isRFPLikely` and added to `firefox/manifest.json` as a content script. Within `common/document_start.js` file, it can thus be checked whether this function exists and if it is run if it does. This approach removes the need to check whether the browser is Firefox via user agent sniffing, as it has been used before.

Input: rfpCompatibilityOn (whether the compatibility mode is already on)
Output: isLikely (Boolean value whether RFP was heuristically detected)
Data: $t_{\textbf{last}}$ (when was this function last executed), done, rfpTests

if done **then**
  return *false*; `// Run at most once per page load`
else if `Date.now()` $- t_{last} < 172\,800\,000$ **then**
  return *false*; `// Run only every two days`
end
$t_{\text{last}} \leftarrow$ `Date.now();` `// Stored in local storage`
if rfpCompatibilityOn **then**
  return *false*; `// Compatibility is already on`
end
$S_{\text{total}} \leftarrow 0$; `// Total score`
$S_{\text{passed}} \leftarrow 0$; `// Passed score`
$\rho \leftarrow 0$; `// Confidence ratio`
foreach test $\in$ rfpTests **do**
  $S_{\text{total}} \leftarrow S_{\text{total}} +$ test.weight;
  passed $\leftarrow$ `test.execute();`
  if passed **then**
    $S_{\text{passed}} \leftarrow S_{\text{passed}} +$ test.weight;
  end
  $\rho \leftarrow S_{\text{passed}}/S_{\text{total}}$;
end
if $\rho \geq \frac{2}{3}$ **then**
  `createNotification();`
  return *true*;
else
  return *false*;
end

Figure 5.7: Algorithm of the heuristic RFP tester.

[5] In some exceptional cases, a test can neither fail nor pass, e.g. when WebGL is disabled, it is impossible to tell whether RFP perturbs its values. In these cases, the `execute()` function returns a `null` value instead of a Boolean value. In this case, the test runner will not add the test's weight to either the passed or the total score.

Should this function determine that RFP is likely enabled without the compatibility mode, it will notify the user via a notification. The `createNotification()` function in the algorithm from Figure 5.7 corresponds to a `browser.runtime.sendMessage()` call that sends a message with a `purpose` value of 'rfp-detected-no-compat'. The file `common/fp_detect_background.js:804` will intercept this runtime message, creating a notification prompting the user to consider enabling compatibility mode in JShelter's settings. A simple notification functions here as well as a proof of concept; however, a more sophisticated and less intrusive informing system could be implemented, such as a section in the JShelter popup page.

The tests themselves are not exposed and contained in the `firefox/rfp_tests.js` file, each as a separate function largely copied (some slightly adjusted) from the RFP-Tester project. Table 5.8 shows the tests included in the `firefox/rfp_tests.js` file.

Test	Accuracy	Reliability	Complexity
~~**Canvas readout noise detection**~~	~~Unknown~~	~~Unknown~~	~~High~~
Navigator properties	High	Medium	Low
~~**Timezone and locale**~~	~~Medium~~	~~Medium~~	~~Low~~
WebGL debug properties	High	Medium	Low
`AudioContext` output latency	Medium	Medium	Low
Enumerating text-to-speech voices	Low	Low	Low
~~**MediaDevices order and types**~~	~~Medium~~	~~Medium~~	~~Medium~~
CSS media query values	Low	Low	Low
Enumerating gamepads	Low	Low	Low
Fetching `NetworkInformation`	Medium	Low	Low

Figure 5.8: Table of tests included in the `firefox/rfp_tests.js` file.

Tests marked with strike-through were excluded from the final implementation for the following reasons:

- Detecting RFP white noise from extracted canvas images could be feasible by determining entropy, but the test was not implemented due to complexity and the need for asynchronicity. Similarly, the `MediaDevices` test would require waiting for device enumeration, and it has therefore not been implemented.
- While implementing the 'Timezone & locale' test, the script was fetching my real time-zone despite the RFP-Tester and direct JavaScript calls showing the Icelandic time-zone (as explained in Section 2.2). The time-zone perturbation may occur later during page load; this needs further research. As it always failed, the test was disabled.
- Enumerating `SpeechSynthesis` voices synchronously makes it possible to assert no voices. An empty voices array, however, may caused by a misconfigured text-to-speech back-end or not having one installed. The main difference RFP-Tester tests for is whether the asynchronous function 'speak' causes an error. Having no text-to-speech back-end would usually cause an error, but RFP makes `speak()` simply do nothing. As this is impossible to implement in a synchronous environment, only the empty voices array is checked.

5.4 Issues and Limitations

The current implementation of the heuristic detection of 'Resist Fingerprinting' (RFP) and adjustment of JShelter's levels to it function as a working proof of concept, and while it resolves the RFP-JShelter overlap, some parts would need more time, consideration or "another look". This section discusses these possible considerations, issues I have encountered while implementing, and possible future changes to consider.

5.4.1 Canvas Noise Exemptions

As described in Section 2.1 and shown in Figure 4.2c, RFP adds heavy white noise to re-extracted canvas images. However, some websites use this feature legitimately. Using an off-screen canvas for side-rendering and then extracting and pasting the result to another visible canvas is a standard approach in optimising complex animations[6]. In these cases, RFP's noise would notably worsen user experience or even lead to site breakage. After some websites were reported to be broken by this perturbation, Firefox now displays a small popup prompt that allows users to add an exception for the current domain. The canvas perturbation is turned off entirely, which may be misused when allowed.

If a user uses JShelter alongside RFP, the JShelter's noise would still be applied. This noise would change the extracted image, though in a way unnoticeable to the user, as shown in Figure 4.2b. In plain terms, the exception is not "all or nothing". Unfortunately, as the RFP compatibility mode turns off JShelter's canvas noise entirely, as shown in Table 4.3, this is no longer be the case.

The ideal resolution to this issue would be accessing the exempted domains or querying an API informing JShelter of this exception, allowing the canvas perturbation to be applied. Unfortunately, Firefox exposes no such API and offers no way to detect such an exception programmatically. One way to detect an exemption could be for JShelter to obtain a canvas readout and measure its entropy. However, this would come at a performance cost, and it is impossible to meaningfully adjust the level after the page has already loaded.

5.4.2 RFP-Exempted Domains

Not only is it possible to create canvas exceptions, but domains can be entirely exempted from 'Resist Fingerprinting' entirely by adding them to `privacy.resistFingerprinting.exemptedDomains`. While the Bugzilla issue regarding this feature[7] suggests one would need to set an experimental flag `privacy.resistFingerprinting.testGranularityMask` to '4', I found that adding the RFP-Tester to the `exemptedDomains` completely turned off all RFP perturbations. Figure 5.9 shows a screenshot of the RFP-Tester not detecting RFP after its domain `*.github.io` was added to `exemptedDomains`.

[6] https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Optimizing_canvas
[7] https://bugzilla.mozilla.org/show_bug.cgi?id=1711619#c4

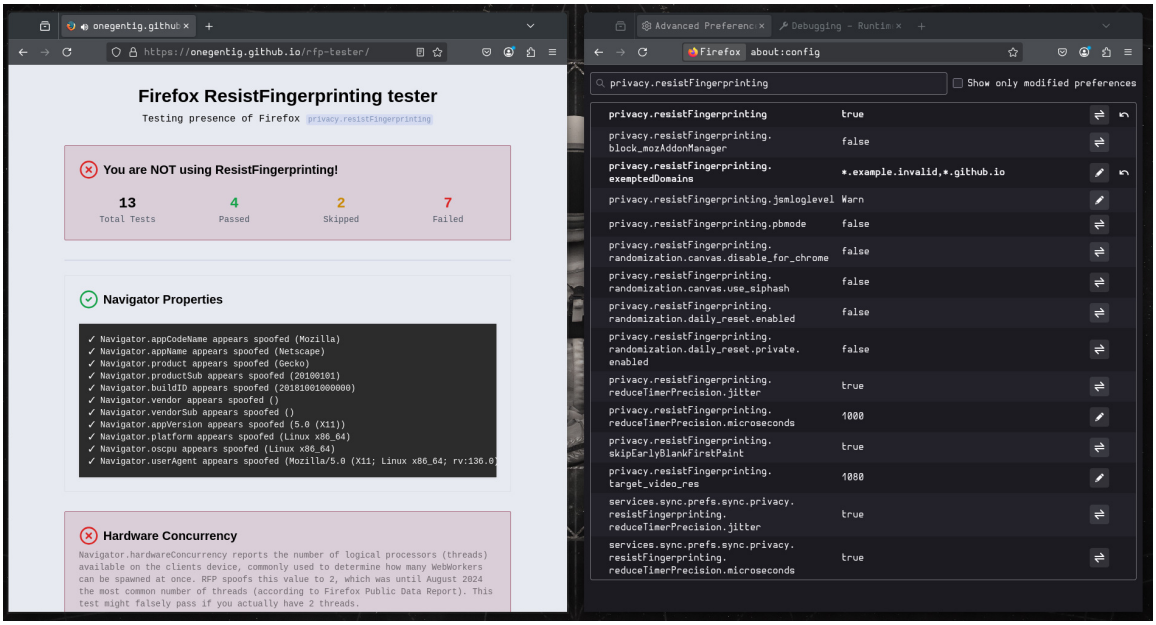


Figure 5.9: Screenshot showing RFP-Tester failing due to its exempted domain despite RFP being visibly enabled.

Similar to the issue described in Section 5.4.1, the ideal solution would be for JShelter to access this browser setting and determine that the current domain is exempted. In such a case, it could disregard RFP entirely and work as if the compatibility mode was not enabled. Unfortunately, Firefox does not expose this setting via any API.

5.4.3 Upcoming Granular Perturbations

The so-called 'Fingerprinting Protection' (FPP) was mentioned in Section 2.7 as a planned feature of Firefox. While the development seems to have stalled, implementing granular fingerprint protection would make this compatibility mode much less helpful. Currently, the compatibility mode considers 'Resist Fingerprinting' (RFP) as a singular setting – adjusting for FPP would require splitting the compatibility mode to check for each perturbation individually and adjusting accordingly per wrapper.

Yet again, Firefox currently does not expose any of the several browser settings related to FPP, making planning ahead or making a future-proof design impossible. In September 2024, a Bugzilla issue was opened, asking to expose one of FPP's settings[8]. After discovering the lack of exposed browser settings and how it made a proper (smarter) RFP adjustment of JShelter impossible, I have added my own comment[9] to the issue, summarising both previous sections. Should the development of FPP continue and eventually the feature becomes mainstream, I hope the `privacy` API would be appropriately amended to account for the setting's granularity. Currently, adjusting for FPP is impossible to do without relying on heuristic "guesswork".

[8] https://bugzilla.mozilla.org/show_bug.cgi?id=1915395
[9] https://bugzilla.mozilla.org/show_bug.cgi?id=1915395#c2

5.4.4 The Invisible Adjustment

The level adjustment to 'Resist Fingerprinting' (RFP) occurs after the level's processing and serialisation into a JSON file through the `getCurrentLevelJSON()` function. While the current method does what it should, the changes are not reflected on the actual level switches seen by the user on the JShelter's popup page. For example, when the user has a 'Strict' (L3) protection level selected, the WebGL protection will display that WebGL's properties are floored and cleared. In reality, the wrapper returns the values set by the RFP's minimum compatibility mode (as described in Section 4.2.2). To make this feature production-ready, it would be necessary to rethink where the adjustment happens so that the changes are reflected in the displayed level but also do not modify the level permanently (i.e. turning RFP or the compatibility mode off reverts any changes the adjustment function has done).

5.4.5 The Notification System

The heuristic detection of RFP is executed every two days if the compatibility mode is not on. While the performance impact should be minimal, the test may come out as a false positive, repeatedly notifying the user of a feature that is not beneficial for them. With the current implementation, the best option for such users would be to enable the compatibility mode – it would simply detect that RFP is off and never do anything. However, some users may use RFP and JShelter together and intentionally want the compatibility mode off.

Removing the heuristic detection and notification may result in many users being unaware of the existence of compatibility mode. As noted in Section 4.4, the 'privacy' permission cannot be enabled in the `firefox/manifest.json` file by default to detect RFP with certainty. One possible option would be to replace the notification with an alert box on JShelter's popup page.

Chapter 6

Testing

The primary goal of this work was to make JShelter adapt its levels when it detects Firefox's 'Resist Fingerprinting' (RFP). Side features included heuristic detection of RFP without the use of `privacy.websites.resistFingerprinting` due to a lack of the 'privacy' permission and adding a new option to the JShelter's options that are removed on a browser without RFP. This section is focused on testing whether the primary goal and the goals of its side features were met and behave as expected.

All tests use the modified version of JShelter 0.20[1] with RFP adjustments implemented, as described in Chapter 5. Unless stated otherwise, the tests are conducted in Firefox 136.0.1 (released on 11th March 2025)[2] with Gecko 0.36[3] on Fedora 41[4] (x86_64 architecture; X11).

6.1 Test 1: Level Adjustment

The main functionality of this addition to JShelter is to change the protection level when RFP is detected. This test determines whether the change is correctly applied on all built-in levels.

For this test, a fresh profile of Firefox 136.0.1 is used, with initially no extensions installed and no preferences changed – RFP and JShelter are added in later setups of this test and are checked to see if the outcomes match expectations. These tests use two tools to assert what changes between the protections – 'RFP-Tester'[5], developed by me as a part of Chapter 2's research, and the web app 'Am I Unique?'[6].

This test will first be executed on Firefox without any protections, then with RFP and JShelter alone, to set a baseline to compare their combinations. In Section 6.1.5, RFP will be tested with JShelter's L2 and in Section 6.1.6 with JShelter's L3.

[1] https://pagure.io/JShelter/webextension/tree/78e2d347a5359eac3569c655b36410b6b0861dbb
[2] https://www.mozilla.org/en-US/firefox/136.0.1/releasenotes
[3] https://github.com/mozilla/geckodriver/releases/tag/v0.36.0
[4] https://docs.fedoraproject.org/en-US/fedora/f41
[5] https://onegentig.github.io/rfp-tester
[6] https://www.amiunique.org/fingerprint

6.1.1 Test 1: No protection

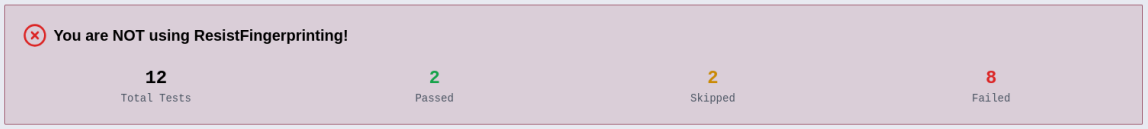


Figure 6.1: RFP-Tester results shown on tested device without any anti-fingerprinting measures.

With no privacy-enhancing tools or changed preferences, the RFP-Tester correctly determined that RFP was not in use. Eight of the 12 total tests failed, while two were skipped and two passed.

Test	Result	Expected Result	Note
Navigator properties	Fail	Fail	
WebGL properties	Fail	Fail	
Web Speech API	Fail	Fail	The tested device had eSpeak TTS back-end installed.
AudioContext output latency	Fail	Fail	
Timestamp rounding and jitter	Fail	Fail	
Time-zone and locale	Fail	Fail	The tested device had a 'Europe/Prague' time-zone.
Screen data	Fail	Fail	
Media Devices	Fail	Fail	The tested device only had one 'audioinput' device.
DNS timing	Pass	Fail	This test likely failed due to the hostname's presence in DNS cache.
CSS queries	Pass	Pass	This test passes on most devices with regular monitors, the tested device included.
Gamepad API	Skip	Skip	This test is skipped by default due to low reliability.
NetworkInformation API	Skip	Skip	This test is skipped by default and only fails if the deprecated API is detected.

Figure 6.2: Table of RFP-Tester results on the tested device without any anti-fingerprinting measures.

The 'Am I Unique?' tool determined that the tested device was still unique in its database. The device was highly unique in having only one media device (0.65 % commonality), a unique user agent string (0.13 %), and a re-extracted image from a WebGL canvas (0.01 %). Overall, the aggregated attributes resulted in a commonality score of 48.95 %. This indicates that while some specific individual attributes are highly unique, the com-

plete fingerprint profile of the device shares approximately half of its characteristics with the broader population in the database.

6.1.2 Test 1: RFP

Figure 6.3: RFP-Tester results shown on tested device with RFP enabled.

As expected, setting `privacy.resistFingerprinting` to `true` has made all assertions of the RFP-Tester correctly pass and detect RFP. Out of the 12 tests, 10 pass, with the remaining two being skipped due to being considered too inconclusive (Gamepad API and deprecated NetworkInformation API).

The uniformity reported by the 'Am I Unique?' web app has improved. With a fake device added by RFP, the media devices commonality percentage rose from 0.65 % to 12.14 %, and WebGL canvas, as it was no longer available, rose from almost 0 % to 14.72 % (now part of an anonymity set with other users with WebGL disabled). The plain canvas test, however, now shows a 0 % match with any of the service's stored samples. The user agent string remained at 0.13 %. The overall aggregated commonality percentage over all tested properties was 50 % (a 0.05 % improvement in uniformity).

6.1.3 Test 1: JShelter L2

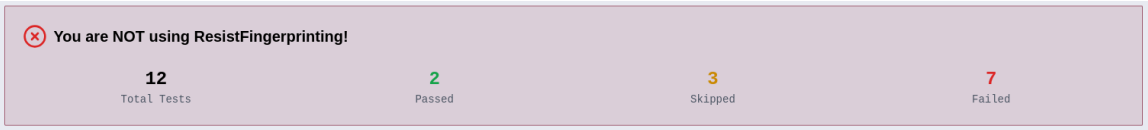


Figure 6.4: RFP-Tester results shown on tested device with JShelter L2.

JShelter, on its 'Recommended' level (L2), was not expected to cause an improvement in the results, compared to having no protection, as shown in Section 6.1.1. After all, the goal of JShelter's L2 is to make the user more unique rather than blend in. Compared to having no protection, the only notable change is that one additional test was skipped. This was the RFP-Tester's 'Time Precision' test, which works by obtaining fifty timestamp samples from `performance.now()` one millisecond apart and measuring each consecutive sample's relative difference (delta). Without RFP, there should be 1 millisecond between each sample (a consistent delta of 1 ms), while RFP applies a consistent jitter of 16.667 ms, as described in Section 2.3. JShelter adds noise to the timestamps without a consistent delta, leading to what the tester calls "inconclusive variance". This result is shown in Figure 6.5.

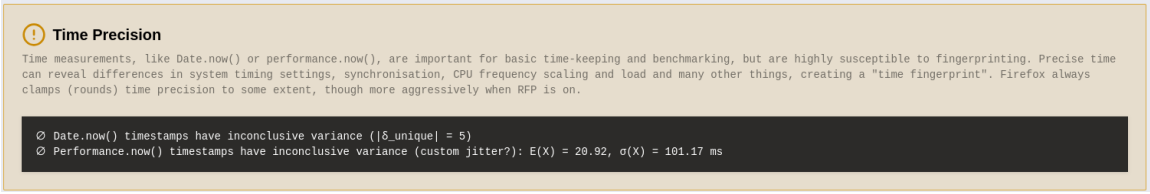


Figure 6.5: RFP-Tester showing 'Time Precision' test being skipped on Test 1, Setup 3.

As for the 'Am I Unique?' test, as expected, the conformity percentage lowered slightly from 48.95 % (no protection) to 47.85 %. The score most notably fell on navigator properties (0 %), WebGL vendor and renderer (both 0 %) and randomised media devices (0 %).

6.1.4 Test 1: JShelter L3

Figure 6.6: RFP-Tester results shown on tested device with JShelter L3.

JShelter, on its 'Strict' level (L3), had a similar result to the previous setup with only one additional test passing. The test 'Navigator Properties' previously failed, as L2 randomised the number of available logical processing units visible via `Navigator.hardwareConcurrency`. L3 sets the value to 2, identically to RFP. One additional passing test is not enough for the tester to believe that RFP is enabled.

The 'Am I Unique?' test run with L3 was notable by the absence of many values, such as no supported video formats (0.13 %), audio formats (0.05 %) or even audio data output support (which is, to my surprise, the case with over 61 % of all service's fingerprints). The aggregated conformity score was 48.91 % – slightly higher than JShelter L2 and lower than RFP; both comparisons were as expected.

6.1.5 Test 1: RFP with JShelter L2

Expectation: L2 focuses on randomising JavaScript properties. The 'Time Precision' test will surely fail, as described in Section 6.1.3, while tests like 'WebGL Properties' or 'Media Devices' may or may not randomly fail. However, even in the worst case – if both these tests fail – the overall test suite should still detect RFP, as three failed tests from the ten counted are still within the RFP-Tester's threshold. Once compatibility mode is enabled, the 'Time Precision' should pass while the two randomisation measures remain unchanged.

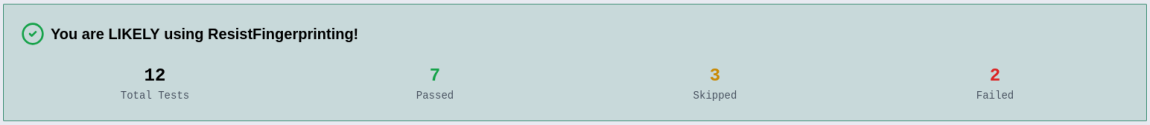


Figure 6.7: RFP-Tester results shown on tested device with RFP and JShelter L2 without compatibility mode.

The test result shown in Figure 6.7 shows this worst outcome. The 'Time Precision' test was skipped due to high variance, as shown in Figure 6.5. Both randomisation perturbations of JShelter – WebGL and media devices – were applied and led to two test failures. Though these tests failed, RFP was still detected.

Figure 6.8: RFP-Tester results shown on tested device with RFP and JShelter L2 with compatibility mode.

After enabling the compatibility mode, the skipped 'Time Precision' test passed, with other tests remaining unaffected. This result matches this setup's expectation and the design decisions noted in Section 4.2. On L2, the time rounding and jitter are delegated to RFP, but the randomisation is kept as-is. The result of this test shows that the level adjustment was correctly applied.

On 'Am I Unique?', the combination of RFP and JShelter L2 without the compatibility mode scored an aggregated commonality of 49.75 %, which is quite close to RFP's 50 %. Compared to L2's original 47.85 %, the combination made the browser less unique. After enabling the compatibility mode, the aggregated commonality rose to 49.9 %, coming even closer to RFP.

6.1.6 Test 1: RFP with JShelter L3

Expectation: JShelter's L3 "delegates" a lot to RFP to make use of its larger anonymity set rather than flooring values. Without the compatibility mode, the result is likely to look similar to L2 with RFP, as shown in Figure 6.7, with floored values rather than randomised ones. With the compatibility mode enabled, most, if not all, tests are likely to pass.

Figure 6.9: RFP-Tester results shown on tested device with RFP and JShelter L3 without compatibility mode.

As expected, the results of L2 with RFP and L3 with RFP look identical, with seven passing, three skipped, and two failing tests. The reason for each test failure is identical to L2's, as described in Section 6.1.5; just the values are floored rather than randomised now.

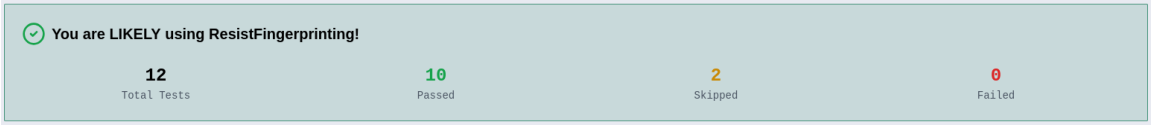


Figure 6.10: RFP-Tester results shown on tested device with RFP and JShelter L3 with compatibility mode.

As shown in Figure 6.10, enabling the compatibility mode worked as expected. The results of the RFP-Tester in this setup are identical to those with RFP alone, as shown in Figure 6.1.

The aggregated commonality score of L3 with RFP by 'Am I Unique?' is also comparable to that of L2 with RFP, scoring 49.51 %. The browser appears slightly more unique than with L2 and RFP with compatibility mode on, though less unique than with L2 and RFP alone. With the compatibility mode enabled, the percentage rose to 50.11 %, slightly outperforming even RFP used alone in terms of uniformity.

6.1.7 Test 1: RFP with a Custom JShelter Level

Expectation: As decided in Section 4.3, level adjustment is only applied to built-in levels. The goal of this test is to prove that custom levels are not modified by the `applyRFPAdjustments()` function. For this test, a custom level was created with these protections:

- High time precision reduction. This should cause the 'Time Precision' test to be skipped without compatibility mode intervening and turning off this perturbation.
- Flooring of WebGL properties. The compatibility mode would turn off this perturbation to make use of WebGL's minimal compatibility mode. However, without it, the values will be set to zeroes or empty strings, failing the 'WebGL Properties' test.
- Remove all media devices. With default permissions, the RFP-Tester will expect at least two media devices, while it shall find none.

The result card of the RFP-Tester is expected to look identical, or at least similar, to L3 with RFP without compatibility mode, as shown in Figure 6.9.

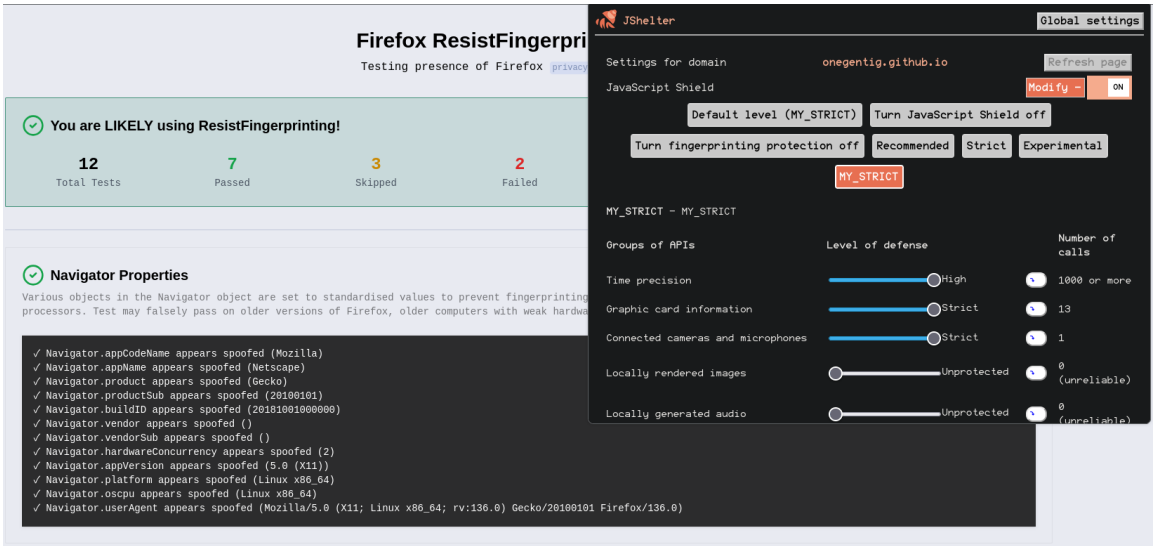


Figure 6.11: RFP-Tester results shown on tested device with RFP and a custom level, with a shown JShelter popup window. A right margin was added by hand for this screenshot.

The resulting count of passed and failed tests being identical to L3 with RFP confirms that the adjustments were not applied as expected.

6.1.8 Test 1: Conclusion

All runs of the RFP-Tester with different JShelter protection levels and RFP settings show that the adjustment function correctly changes the levels if RFP is detected. This change was reflected in passed assertions of the test suite as well as a change in conformity score reported by the 'Am I Unique?' web app. A summary table of averaged conformity percentages is shown in Table 6.12. The test exhibited in Section 6.1.7 further confirmed that the RFP adjustments are not applied on custom levels as intended.

	No RFP	**RFP**	**RFP + compat.**
None/L0	48.95 %	50 %	N/A
L2	47.85 %	49.75 %	49.90 %
L3	48.91 %	49.51 %	50.11 %

Figure 6.12: Table of averaged conformity scores reported by 'Am I Unique?' web app.

6.2 Test 2: Heuristic Detection

The secondary feature implemented into JShelter was the heuristic detection of Resist Fingerprinting (RFP), as described in Section 5.3. This test aims to verify that it correctly detects RFP when on and shows a notification or does no action when RFP is off. This test should also verify that the test suite is not detected by JShelter's Fingerprinting Detector (FPD).

The testing environment is unchanged from Test 1, described in Section 6.1. For this test, I will be using the website `https://www.stud.fit.vutbr.cz`. This website has no JavaScript code; hence, anything detected by FPD will be caused by the test suite. The desired outcome is that FPD lists no fingerprinting calls.

6.2.1 Test 2: Privileged Context Problem

The initial run of this test revealed an oversight in the scheduling implementation. During testing, the extension was being loaded into Firefox via the `about:debugging` window. The content script `document_start.js`, and thusly `isRFPLikely()`, was executed immediately after installation. Regardless of RFP being enabled, as `about:` is a privileged scheme[7], the test would fail and not re-run for the next two days. Some users would never be notified of the compatibility mode, especially if they use `about:blank` as their home page. Privileged contexts were mentioned in Section 5.3 when deciding when to run the test suite. However, I did not account for `document_start.js` running in such a context.

To resolve this issue, I updated the detection function to verify privileged contexts prior to attempting the detection. The added conditional block is shown in Figure 6.13.

[7] `https://searchfox.org/mozilla-central/rev/1f56e3062b8e2f71aad6679d48066ae7733cb855/dom/base/nsContentUtils.cpp#2450`

```
function isRFPLikely (rfpInfo) {
    /* ... prevention of multiple runs per page load ... */

    if (
        window.location.protocol === 'chrome:' ||
        window.location.protocol === 'resource:' ||
        window.location.protocol === 'view-source:' ||
        window.location.protocol === 'about:' ||
        window.location.protocol === 'moz-extension:' ||
        window.location.hostname === 'addons.mozilla.org'
    ) {
        return false;
    }

    /* ... other pre-run checks ... */
    /* ... the rest of the function ... */
}
```

Figure 6.13: Adjustment to `isRFPLikely()` function to account for privileged contexts.

6.2.2 Test 2: Result

After fixing the issue with privileged contexts, I conducted the test again. Initially, I loaded the modified JShelter with RFP turned off and refreshed the testing website. No notification was shown as expected. A hint of the test running could be noticed in the browser console – besides existing logs from `document_start.js`, a warning about automatically starting `AudioContext` is displayed, as shown in figure 6.14. The output latency test causes this. As `AudioContext` is only instantiated to check for latency, not for actual use, this warning is of no concern.

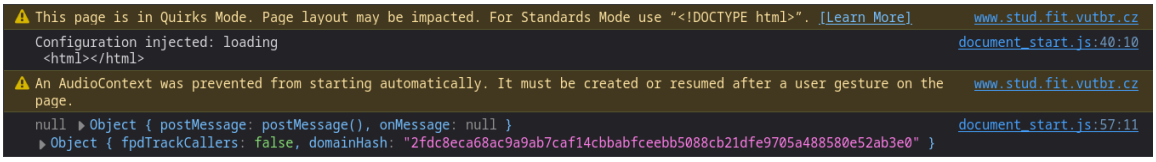


Figure 6.14: Firefox Console after loading `document_start.js`.

As the RFP check is done once every two days, I have reinstalled the modified JShelter extension and enabled `privacy.resistFingerprinting` in `about:config` to test for the positive tester outcome. As I opened the testing website, the expected notification was displayed, informing me of having RFP without compatibility mode. The way the notification appeared on my testing device is shown in Figure 42.9.

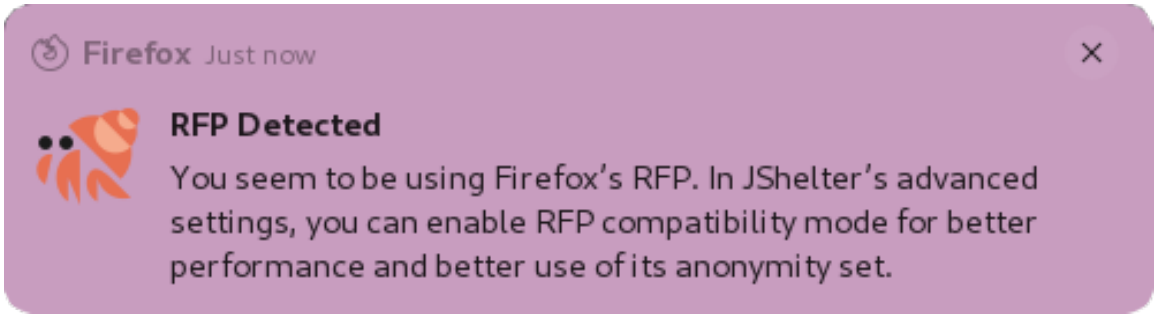


Figure 6.15: 'RFP Detected' notification as displayed on modified GNOME 47.

The final check of this test was whether JShelter's FPD module had flagged the tester as a fingerprinter. As the test is run synchronously and before FPD is loaded, it is expected that the fingerprinting report will be empty. Figure 42.9 shows this fingerprinting report from the testing site along with the notification.

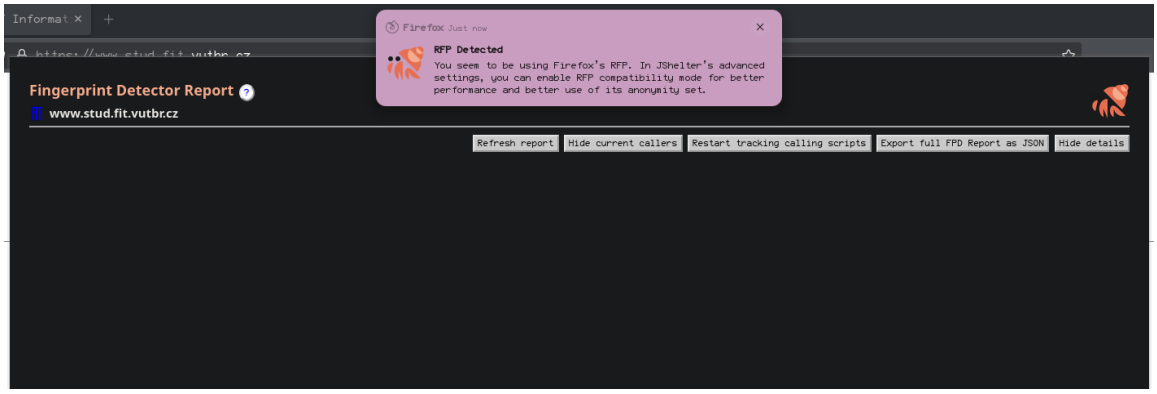


Figure 6.16: Empty FPD report shown after the execution of `isRFPLikely()`.

In conclusion, this test confirmed that the heuristic detection detects RFP, shows a notification and does not cause issues with FPD.

Chapter 7

Conclusion

It has been said many times before that the current internet landscape is not very privacy-friendly. Modern convenience often comes at the cost of privacy, and things are not likely to change any time soon [5]. For the growing number of people who are aware of and unhappy about this trade-off, there are various privacy-enhancing tools to reduce one's internet fingerprint [21, 6]. Extensions like JShelter or browser settings like 'Resist Fingerprinting' (RFP) are examples of such tools. This thesis took a look at how they work together, identified some points of conflict or redundancy, and attempted to resolve them.

Investigating the interaction between RFP and JShelter began with a comprehensive analysis of RFP's fingerprinting countermeasures in Chapter 2, documenting the specific browser properties it modifies and other perturbations it employs. This was followed by a more focused examination of JShelter's protections applied in a Firefox (Gecko) environment in Chapter 3, focusing on areas of potential overlap between the two. Based on these analyses, Chapter 4 analysed their interactions from a theoretical and practical standpoint and proposes a compatibility mode for JShelter, adjusting the protection levels to avoid redundancy while maintaining comprehensive fingerprinting protection. After identifying an issue with missing permission, a heuristic detection of RFP was proposed to inform the user of this setting. The compatibility mode and the heuristic detection were implemented on JShelter 0.20, as detailed in Chapter 5, and their efficacy was tested in Chapter 6. As testing demonstrated, the compatibility mode successfully adjusts the built-in protection levels and accurately detects RFP via a built-in test suite.

Despite the successful implementation of the compatibility mode, several limitations have prevented this feature from being presented to and potentially becoming a part of the JShelter extension. These issues are described in Section 5.4. Most notably, the adjustment happens within the level cache, making the adjustment invisible "magic" in the settings. Furthermore, the inability to detect exceptions (especially for the canvas noise) would silently compromise some fingerprinting countermeasures.

However, as the adjustment was successful in the end, after resolving the noted issues, rethinking some of the feature's specifics and further testing, this feature could be beneficial to the JShelter extension in removing redundancy and improving uniformity in JShelter's 'Strict' protection mode. Future work that could extend and improve upon this thesis could also involve better user experience, especially in terms of the heuristic detection notification, improved detection, performance optimisation and testing, and feature-level granularity, which could become essential if Mozilla should continue with the development of a more granular 'Fingerprinting Resistance' (FPD).

Bibliography

[1] *JShelter*. online. Available at: https://jshelter.org/.

[2] Andalibi, V.; Christophe, F. and Mikkonen, T. Analysis of Paradoxes in Fingerprint Countermeasures. In: *Proceedings of the 21st Conference of Open Innovations Association*. Helsinki, Finland: FRUCT, November 2017, p. 398–401. Available at: https://researchportal.helsinki.fi/files/98530323/And.pdf.

[3] Bajanik, M. *Why Anti-Fingerprinting Techniques Don't Work in Browsers*. online. FingerprintJS Inc, september 2022. Available at: https://fingerprint.com/blog/browser-anti-fingerprinting-techniques/.

[4] Borking, J. J.; Verhaar, P.; Eck, B. M. A. van; Siepel, P. et al. PET. In: Blarkom, G. W. van; Borking, J. J. and Olk, J. G. E., ed. *Handbook of Privacy and Privacy-Enhancing Technologies: The case of Intelligent Software Agents*. The Hague, Netherlands: College Bescherming Persoonsgegevens, November 2003, chap. 3, p. 33–54. ISBN 90-74087-33-7. Available at: https://www.andrewpatrick.ca/pisa/handbook/Handbook_Privacy_and_PET_final.pdf.

[5] Cyril, H. *Are We Sacrificing Privacy for Convenience in the Digital Age?*. online. TechBullion, november 2023. Available at: https://techbullion.com/are-we-sacrificing-privacy-for-convenience-in-the-digital-age.

[6] European Commission. *Flash Eurobarometer 443: e-Privacy*. European Commission, Directorate-General for Communications Networks, Content & Technology (DG CONNECT), december 2016. Available at: https://europa.eu/eurobarometer/surveys/detail/2124.

[7] Flanagan, D. *JavaScript: The Definitive Guide*. 6th ed. Sebastopol, CA, USA: O'Reilly Media, march 2011. ISBN 9780596805524.

[8] Ghayour, F. and Cantor, D. *Real-Time 3D Graphics with WebGL 2: Build interactive 3D applications with JavaScript and WebGL 2 (OpenGL ES 3.0)*. 2nd ed. Birmingham, United Kingdom: Packt Publishing, october 2018. ISBN 781788629690.

[9] Hayes, B. Computing Science: Uniquely Me!. *American Scientist*, april 2014, vol. 102, no. 2, p. 106–109. ISSN 0003-0996. Available at: http://www.jstor.org/stable/43707158.

[10] Hraška, P. *Browser Fingerprinting*. Bratislava, Slovakia, 2018. Master's thesis. Comenius University. Supervisor Forišek, M. Available at: http://www.dcs.fmph.uniba.sk/diplomovky/obhajene/getfile.php/hraska_diploma_thesis.pdf?id=434&fid=771&type=application/pdf.

[11] International Telecommunication Union. *State of digital development and trends in the Europe region: Challenges and opportunities.* Geneva, Switzerland: ITU Publications, february 2025. Available at: https://www.itu.int/itu-d/reports/statistics/wp-content/uploads/sites/5/2025/02/ITU_SDDT_Europe.pdf.

[12] Kocher, P.; Horn, J.; Fogh, A.; Genkin, D.; Gruss, D. et al. Spectre Attacks: Exploiting Speculative Execution. In: *2019 IEEE Symposium on Security and Privacy (SP).* San Francisco, CA, USA: IEEE Computer Society, 2019, p. 1–19.

[13] Mowery, K.; Bogenreif, D.; Yilek, S. and Shacham, H. Fingerprinting information in JavaScript implementations. In: Wang, H., ed. *Proceedings of W2SP 2011.* Oakland, CA, USA: IEEE Computer Society, May 2011.

[14] Mowery, K. and Shacham, H. Pixel Perfect: Fingerprinting Canvas in HTML5. In: *Proceedings of W2SP 2012.* San Francisco, CA, USA: IEEE Computer Society, May 2012.

[15] Mozilla Developer Network. *Browser detection using the user agent string (UA sniffing).* online. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Browser_detection_using_the_user_agent.

[16] Mozilla Support Staff. *Enhanced Tracking Protection in Firefox for Desktop.* online. Mozilla. Available at: https://support.mozilla.org/kb/enhanced-tracking-protection-firefox-desktop. [cit. 2024-12-19].

[17] Mozilla Support Staff. *Resist Fingerprinting.* online. Mozilla. Available at: https://support.mozilla.org/kb/resist-fingerprinting. [cit. 2024-12-19].

[18] Mozilla Wiki Contributors. *Security/Fingerprinting.* online. Mozilla. Available at: https://wiki.mozilla.org/Security/Fingerprinting. [cit. 2024-11-18].

[19] Mozilla Wiki Contributors. *Security/Tor Uplift.* online. Mozilla. Available at: https://wiki.mozilla.org/Security/Tor_Uplift. [cit. 2024-09-19].

[20] Murdoch, S. Hot or Not: Revealing Hidden Services by their Clock Skew. In: *CCS '06: Proceedings of the 13th ACM conference on Computer and Communications Security.* Alexandria, VA, USA: ACM Press, October 2006, p. 27–36.

[21] Pew Research Center. *The state of privacy in post-Snowden America.* online. May 2016. Available at: https://www.pewresearch.org/short-reads/2016/09/21/the-state-of-privacy-in-america/.

[22] Pfitzmann, A. and Hansen, M. *A Terminology for Talking about Privacy by Data Minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management.* online. Technical document. Dresden, Germany, august 2010. Available at: http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf. V0.34.

[23] Polčák, L.; Saloň, M.; Maone, G.; Hranický, R. and McMahon, M. JShelter: Give Me My Browser Back. In: INSTICC. *Proceedings of the 20th International Conference on Security and Cryptography (SECRYPT).* Rome, Italy: SciTePress, 2023, p. 287–294. ISBN 978-989-758-666-8.

[24] Rokicki, T.; Maurice, C. and Laperdrix, P. SoK: In Search of Lost Time: A Review of JavaScript Timers in Browsers. In: *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. Vienna, Austria: IEEE Computer Society, 2021, p. 472–486.

[25] Saloň, M. *Detekce metod zjišťujících otisk prohlížeče.* Brno, Czechia, 2021. Master's thesis. Brno University of Technology. Supervisor Polčák, L. Available at: https://www.fit.vut.cz/study/thesis/23645.

[26] Statista Search Department. *Average daily time spent using the internet by online users worldwide from 3rd quarter 2015 to 3rd quarter 2024.* Statista, february 2025. Available at: https://www.statista.com/statistics/1380282.

[27] W3C Privacy Working Group. *Mitigating Browser Fingerprinting in Web Specifications.* online. World Wide Web Consortium (W3C). Available at: https://w3c.github.io/fingerprinting-guidance/.

Appendix A

Contents of the included storage media

The following directory tree describes the hierarchy and the contents of the SD card attached to this thesis. The SD card is in FAT32 file format.

```
/ ..... Root directory of the SD card
├── jshelter ..... Source code of the modified JShelter
│   └── Makefile ..... Makefile to build the modified JShelter
├── jshelter.tar.gz .Tarball of the modified JShelter source code (backup)
├── tester ..... Source code of the RFP-Tester
├── thesis-src ..... LaTeX source code for this thesis
│   ├── assets ..... Images used in this thesis
│   └── bib-style ..... BibTeX styles used in this thesis
└── thesis.pdf ..... This built final thesis
```

Figure A.1: Contents of the attached SD card.