VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

PŘENOS BEZPEČNOSTNÍCH OPATŘENÍ Z PROHLÍŽEČE BRAVE DO ROZŠÍŘENÍ JAVASCRIPT RESTRICTOR

PORTING OF BRAVE FINGERPRINTING PROTECTION TO JAVASCRIPT RESTRICTOR

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MATÚŠ ŠVANCÁR**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. LIBOR POLČÁK, PhD.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Švancár Matúš, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Vývoj aplikací
Název: **Přenos bezpečnostních opatření z prohlížeče Brave do rozšíření JavaScript Restrictor**
Porting of Brave Fingerprinting Protection to JavaScript Restrictor
Kategorie: Web

Zadání:

1. Seznamte se s projektem JavaScript Restrictor.
2. Analyzujte bezpečnostní opatření a ochranu proti získávání otisku prohlížeče implementovanou prohlížečem Brave.
3. Navrhněte postup přenosu opatření z prohlížeče Brave do nástroje JavaScript Restrictor.
4. Přeneste bezpečnostní opatření, pište dostatečně kvalitní a komentovaný kód, aby byl vedoucím práce akceptován do nástroje JavaScript Restrictor.
5. Implementaci otestujte.
6. Práci vyhodnoťte a navrhněte možná zlepšení.

Literatura:

- LAPERDRIX Pierre, BAUDRY Benoit, MISHRA Vikas. *FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques*. In 9th International Symposium on Engineering Secure Software and Systems, 2017.
- NIKIFORAKIS Nick, JOOSEN Wouter, LIVSHITS Benjamin. *PriVaricator: Deceiving Fingerprinters with Little White Lies*. In Proceedings of the 24th International Conference on World Wide Web, 2015.
- SCHWARZ Michael, LACKNER Florian, GRUSS Daniel. *JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits*. In Network and Distributed Systems Security (NDSS) Symposium. 2019.
- HORŇÁK, Peter. *Přenos bezpečnostních opatření z Chrome Zero do JavaScript Restrictor*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Polčák Libor, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 22. října 2020

Abstrakt

Používatelia internetových prehliadačov sú neustále sledovaní, a to bez ich súhlasu. Využitím JavaScript rozhraní je možné získať rôzne informácie o prehliadači, ktoré spolu tvoria odtlačok prehliadača, ktorý je možno následne zneužiť. Preto je cieľom tejto práce využiť robustné riešenie ochrany pred snímaním odtlačku prehliadaču Brave a preniesť ho do rozšírenia JavaScript Restrictor. V rámci tejto práce je analyzovaná problematika získavania odtlačku, ochrana v prehliadači Brave a jej porovnanie s momentálnou ochranou v rozšírení JSR. Je prezentovaný návrh prenosu protiopatrení a následne je popísaný postup implementácie týchto prvkov do rozšírenia prehliadaču. Výslená implementácia bola testovaná a vyhodnotená, pričom sa nová ochrana javí ako účinná.

Abstract

Users of internet browsers are constantly monitored, without their consent. By using the JavaScript APIs, it is possible to obtain various information about the browser, which together form a browser fingerprint, which can then be misused. Therefore, the goal of this work is to use a robust fingerprint protection solution of Brave browser and port it to the JavaScript Restrictor extension. In this work, the problematics of obtaining an fingerprint and countermeasures in the Brave browser are analyzed and then compared with the current protection in the JSR extension. The method of porting of Brave's countermeasures is presented and subsequently the procedure of implementation of these defense elements into the browser extension is described. The resulting implementation has been tested and evaluated, with the new protection appearing to be effective.

Kľúčové slová

odtlačok prehliadača, browser fingerprinting, Brave, Farbling, JSR, JavaScript Restrictor, Javascript, PriVaricator, FPRandom, znáhodnený odtlačok, ochrana proti snímaniu odtlačku

Keywords

browser fingerprinting, Brave, Farbling, JSR, JavaScript Restrictor, Javascript, PriVaricator, FPRandom, randomized fingerprint

Citácia

ŠVANCÁR, Matúš. *Přenos bezpečnostních opatření z prohlížeče Brave do rozšíření JavaScript Restrictor*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Libor Polčák, PhD.

Přenos bezpečnostních opatření z prohlížeče Brave do rozšíření JavaScript Restrictor

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne, pod vedením pána Libora Polčáka. Uviedol som všetky publikácie a literárne pramene, z ktorých som čerpal.

.....
Matúš Švancár
18. mája 2021

Poďakovanie

Chcel by som sa poďakovať vedúcemu práce, Ing. Liborovi Polčákovi, Ph.D. za cenné rady a návrhy, odborné konzultácie a tiež vecné pripomienky pri písaní tejto práce.

Obsah

Kapitola 1

Úvod

Používanie nástrojov na sledovanie používateľov na internete je jedným z problémov posledných rokov. Reklamy a sledovače dali vzniknúť nástrojom na ich efektívne blokovanie, no spôsoby sledovania sa menia. Snímanie odtlačku je spôsob zaznamenávania charakteristík konkrétneho prehliadača, čo môže byť následne zneužité bez vedomia používateľa. Preto vznikajú nástroje a prehliadače, ktoré sa takýmto pokusom zaznamenávať odtlačky a následne ich zneužívať bránia. Prehliadač Brave takúto ochranu obsahuje a táto práca ju rozoberá a popisuje možnosti jej prenosu do rozšírenia JavaScript Restrictor, čo by poskytlo lepšiu ochranu aj používateľom bežných prehliadačov.

Práca je členená do logických celkov a to nasledovne. Kapitola 2 bližšie popisuje fenomén snímania odtlačku prehliadača, motiváciu, či možnosti zneužitia, konkrétne spôsoby získavania odtlačkov a ich rozšírenosť. Nasleduje kapitola 3, kde sú popísané spôsoby ochrany pred snímaním odtlačku prehliadača, konkrétne teoretické, či reálne riešenia aplikované do praxe a ochranu v prehliadači Brave a rozšírení JavaScript Restrictor. Kapitola 4 porovnáva momentálnu ochranu v rozšírení JavaScript Restrictor s ochranou v Brave, a popisuje návrh a stav prenosu opatrení z prehliadača Brave do JSR.

Implementácia je popísaná v kapitole 5. Táto kapitola zhŕňa proces a detaily prenosu opatrení z Brave do rozšírenia prehliadaču. Tiež sú tu popísané jednotlivé zmeny, ktoré boli vykonané a nové veci pridané do rozšírenia.

Nakoniec kapitola 6 zhŕňa postup testovania počas vývoja, ale aj testovanie finálnej verzie ochrany na službách vykonávajúcich snímanie odtlačku prehliadaču. Na konci tejto kapitoly sú tiež vyhodnotené výsledky ochrany a jej zahrnutie do vydania rozšírenia. Sú tiež prezentované smery, ktorými by sa mohol uberať budúci vývoj, nadväzujúci na túto prácu.

Kapitola 2

Browser Fingerprinting

Anglický výraz fingerprinting možno preložiť ako „snímanie odtlačku prsta“, podobne ako pomocou odtlačku prsta je možné identifikovať človeka, ktorému odtlačok patrí, tak aj pomocou odtlačku prehliadača je možné identifikovať konkrétny prehliadač, alebo množinu prehliadačov, ak odtlačok nie je jedinečný. Výrazom „browser fingerprinting“ sa teda myslí „snímanie odtlačku prehliadača“. Pri snímaní odtlačku prehliadača sa využívajú najčastejšie dáta získané z HTTP hlavičiek a JavaScript API (Application programming interface).

Fingerprinting skripty teda zbierajú údaje z HTTP hlavičiek a API prehliadaču a vytvárajú na ich základe dátovú štruktúru – odtlačok . Jednotlivé časti odtlačku sa väčšinou prevádzajú do podoby hashu a navzájom kombinujú aby výsledný odtlačok poskytoval čo najviac identifikujúcich bitov. Narozdiel od cookies, je vytváranie odtlačku prehliadaču na strane klienta bezstavové, čo znamená, že používateľ má iba minimálnu kontrolu nad jeho priebehom a nemá prístup k svojim odtlačkom, ktoré sa ukladajú na serveroch. Taktiež je náročné vôbec detegovať, že k snímaniu dochádza. Používateľ sa však môže pokúsiť chrániť používaním prehliadaču, či rozšírenia prehliadaču, ktoré ponúkajú ich vlastné spôsoby ochrany pred snímaním odtlačku prehliadača. Je vhodné poznamenať, že implementácia úplnej ochrany nie je reálna, a momentálne riešenia majú rôznu účinnosť a rôzny dopad na používateľský zážitok.

Aj keď Eckersley [18] a Laperdrix et al. [28] vypovedali o viac ako 80% jedinečnosti zaznamenaných odtlačkov, treba poznamenať, že dáta môžeme považovať za skreslené. V čase výskumov sa používal Flash a iné pluginy, čo spôsobovalo bezpečnostné medzery a zjednodušovalo napríklad získavanie informácií o fontoch. Tiež je patrné, že dátová sada nereprezentovala väčšinovú populáciu, ale ľudí, ktorí sa o bezpečnosť zaujímajú a navštívili ich stránky, svedčí o tom aj zloženie OS používateľov, pričom Linux bol 10x bežnejší ako v skutočnosti.

Publikácia Gómez-Boix et al. [24] však reprezentuje skutočnú situáciu presnejšie, keďže autori získavali dáta z verejných stránok, od všetkých užívateľov, ktorí súhlasili s cookies. Odhliadnuc od faktu, že sa jednalo o francúzske weby, čiže časová zóna a jazyk boli pomerne homogénne, sa jedná o dobrú testovaciu vzorku používateľov. Tiež používali font probing založený na JavaScripte a canvas API, ktoré je v súčasnosti populárne medzi fingerprinting skriptmi. Ich výsledky sú v rozpore s predošlými výskumami a hovoria o 35.7% jedinečných odtlačkov na desktopových počítačoch a dokonca 18.5% na mobilných zariadeniach - výskum však vypovedá aj o tom, že malé zmeny, ako napríklad odlišná časová zóna, či jazyk, by mohli viesť k identifikovateľnosti aj medzi homogénnymi skupinami odtlačkov. Dáta prezentované v článku hovoria aj o celkovom rozdelení meraných prehliadačov do podobnostných množín, zaujímavé je, že počet prehliadačov spadajúcich do malých množín (2-50)

bol typicky najmenší a teda buď bol prehliadač presne identifikovateľný, alebo bol podobný väčšiemu počtu iných prehliadačov. Menšia unikátnosť odtlačkov však nutne neznamená, že používatelia sú v bezpečí, výskumy [28, 30] totiž naznačujú, že počet stránok, ktoré snímajú odtlačok prehliadača stále stúpa. Kým totiž budú prehliadače používať JavaScript API nie je možné sa zbierania odtlačkov úplne zbaviť, je možno sa im však brániť.

2.1 Využitie

Zaznamenané odtlačky je možné použiť na rôzne účely, najčastejšie sa používajú v kombinácii s inými faktormi na sledovanie používateľov. Ak používajú rôzne domény, ktoré používateľ navštevuje, fingeprinting skripty, je možné (v závislosti od jedinečnosti odtlačku) prehliadač ihneď identifikovať. Aj keď je konkrétny prehliadač identifikovaný, nie je jednoduché spojiť s ho konkrétnym používateľom, ale je to možné ak sú informácie o používateľovi získané z formulárov, prihlasovania a podobne. Tiež možno odtlačok priradený k prehliadaču použiť na zacielenie reklamy, sledovanie aktivity prehliadača a podobne. Boli zaznamenané aj skripty, ktoré na základe odtlačku obnovujú zmazané cookies [30]. Riziko okrem sledovania spočíva aj v možnosti bližšie identifikovať používateľovo zariadenie, alebo softvér, a využiť to na zlepšenie šancí útočníka na úspech za pomoci odhalenia slabín konkrétneho zariadenia. Takéto slabiny sa dajú využiť napríklad na Zero day útoky [7].

Fingerprinting je však možné použiť aj na zvýšenie bezpečnosti na webe. Stránky ako Coinbase[1] využívajú fingerprinting na zistenie neoprávnených pokusov o prihlásenie [21, 32]. Na základe odtlačku majiteľa účtu teda zistia pokus o prihlásenie iným prehliadačom a varujú majiteľa o podozrivej aktivite – keďže sa jedná o stránku pracujúcu s finančnými prostriedkami je takáto ochrana vítaná. Podobné využitie snímania odtlačkov dokonca schvaľuje a odporúča WP29 [3], viď use case 7.4 a 7.5.

Existujú aj komerčne dostupné fingerprinting riešenia, ktoré majú stránkam poskytnúť spomínanú funkcionalitu. Príkladom môže byť veľmi populárne fingerprint.js[2], ktoré poskytuje open source knižnicu, ktorú môžu vývojári použiť na vytvorenie vlastnej databázy odtlačkov na svojich stránkach. Taktiež ale ponúkajú spoplatnený program, ktorý umožňuje využívať ich API, ktoré poskytuje rozšírenú funkcionalitu a presnejšiu identifikáciu prehliadaču - nielen na základe odtlačku. Táto služba potom zaisťuje ochranu pred zdieľaním účtov, platobnými podvodmi, ukradnutím účtov a podobne.

2.2 Typy browser FP

Spôsoby získavania odtlačku sa v priebehu rokov menili, najmä kvôli pokusom o opravenie bezpečnostných rizík a ochranu pred zbieraním odtlačku, ale aj kvôli vývinu prehliadačov, jazykov a API. Ďalšie podkapitoly popisujú dnes populárne spôsoby získavania odtlačku, ako aj spôsoby, ktoré sú relevantné vzhľadom na ochranu v prehliadači Brave.

2.2.1 Aktívne vs pasívne získavanie odtlačku

Spôsoby snímania odtlačku prehliadaču sa rozdeľujú podľa spôsobu získavania informácií na aktívny a pasívny fingerprinting [17].

[1] https://www.coinbase.com/
[2] https://fingerprintjs.com/

Pasívny fingerprinting je taký, ktorý na strane klienta nevykonáva žiaden kód a využíva informácie zo sieťových paketov - teda IP adresu, operačný systém, jazyk a podobne. Poskytuje tak menej informácií, na základe ktorých väčšinou prehliadač nejde priamo identifikovať.

Naproti tomu aktívny fingerprinting vykonáva úkony na strane klienta. Využíva sa najmä JavaScript na prístup k rozhraniam, pomocou ktorých sa dajú získať rôzne identifikujúce údaje, ako napríklad rozmery okna, zoznam pluginov, či niektoré hardvérové podrobnosti a podobne. Dôležité je, že tento spôsob je aspoň teoreticky rozpoznateľný na strane klienta.

2.2.2 Cross browser fingerprinting

Je súhrn spôsobov získavania odtlačku, ktorý nie je obrazom konkrétneho prehliadaču, ale používateľovho zariadenia. Vo výsledku je teda podľa takéhoto odtlačku možné používateľa identifikovať, aj keď používa rôzne prehliadače. Na vytvorenie takéhoto odtlačku je možno použiť také údaje, ktoré nezávisia iba od prehliadaču, napríklad rozlíšenie obrazovky, zoznam fontov, informácie z HTTP hlavičiek a podobne [8]. Takéto odtlačky môžu byť nestabilné, napríklad nainštalovanie nových fontov, či aktualizácia prehliadača vedú k zmenám odtlačku. Je však možné použiť heuristiky na identifikáciu zmien a spojenie používateľa s novým odtlačkom. Novšie prístupy, ako napríklad použitie WebGL odtlačku sa ukazujú ako sľubné doplnenie iných údajov pre presnejší cross-browser odtlačok [15].

2.2.3 Canvas odtlačok

Jedna z novších techník získavania odtlačku. Z HTML elementu canvas, ktorý zobrazuje renderované obrázky je možné vyrendrovať testovací text, či tvary, ktoré používateľ nevidí. Obsah obrázku je potom prevedený na hash, ktorý je potom použitý na idektifikáciu. Pri canvase je možno použiť viac faktorov na zvýšenie konečnej entropie. Používajú sa napríklad rôzne efekty priehľadnosti, kombinácia s gradientami, akcelerácia na grafickej karte a podobne. Použitie fallback (font, ktorý sa použije, ak je nastavený na zariadení nedostupný font) a bežného fontu odhalí informácie o OS a nainštalovaných fontoch. Pridaním emoji je možné rozlíšiť platformu, pretože sa vyrendrovaný emoji na rôznych platformách líši. Na rendrovanie textu a tvarov má vplyv hardvér aj softvér, jedná sa teda o komplexný proces, ktorého výsledok závisí na viacerých vrstvách[28], takže ani analýza nie je jendoduchá. Spojením všetkých spomínaných faktorov má výsledný odtlačok veľmi vysokú entropiu. Podľa informácií v [20] sa používanie canvas FP od minulých štúdií zvýšilo a z 10000 top Alexa stránok 7% používa canvas FP a 2.5% canvas based font FP.

Obr. 2.1: Príklad obrázka vygenerovaného pri Canvas FP

2.2.4 WebGL odtlačok

Z WebGL API možno priamo zistiť výrobcu grafickej karty a rendering engine a iné parametre. Tiež sa podobne ako pri Canvas FP používa canvas html element, no pomocou WebGL sa použije na vyrendrovanie viacerých úloh. Bežne sa používa niekoľko textúr, rôzne

nastavenie osvetlenia, či zapnutie funkcií ako anti-aliasing. Výber správnych scén je kľúčové k vytvoreniu kvalitného odtlačku. Aj keď Laperdrix et al. [28] po testovaní vyvodili záver, že WebGL odtlačok je príliš nestabilný na dostatočnú spoľahlivosť, neskôr však Cao et al. [15] hovorí o opaku. Bol použitý iný prístup a vstupné premenné boli striktnejšie obmedzené, čo znížilo nestabilitu a autori dosiahli v kombinácii s inými FP parametrami identifikovateľnosť prehliadaču viac ako 99%. Výsledný WebGL odtlačok potom veľmi závisel od operačného systému a hardvérových komponentov, preto autori spomínaného článku vytvorili algoritmus, ktorý vytvára cross-browser fingerprint masku, ktorú je možné použiť na rozlíšenie konkrétnych zariadení. Podľa článku, boli schopní dosiahnuť identifikovateľnosť až 83%, čo je veľké zlepšenie oproti [8], kde bola dosiahnutá pri cross-browser FP identifikovateľnosť 69% s nižšou stabilitou.

Tabuľka 2.1: príklad WebGL odtlačku
* hodnoty pre WebGL Paramaters sú kvôli veľkému rozsahu skrátené do popisu

WebGL Vendor	Google Inc.
WebGL Renderer	Google SwiftShader
WebGL Data	
WebGL Parameters *	27 rôznych rozšírení 25 rôznych parametrov 36 rôznych shader presností

2.2.5 Audio odtlačok

Ďalším konceptuálne podobným spôsobom snímania odtlačku prehliadaču je analýza zvuku. Jedná sa o jeden z novších spôsobov a aj keď je menej populárny, postupne sa dostáva aj do populárnych FP nástrojov, ako napríklad fingerprintjs[3]. Jednak sa z Web Audio API dá spraviť odtlačok analýzou dostupných objektov a ich parametrov, ako možno vidieť v Tabuľke 2.2, čo síce samo o sebe nemusí byť unikátne, ale poskytne to pri zahrnutí do väčšieho odtlačku niekoľko identifikujúcich bitov. Web Audio API tiež obsahuje rozhrania, pomocou ktorých sa dá načítať, či vytvoriť nový zdroj zvuku, tento signál sa dá potom ďalej modifikovať a meniť pomocou dostupných metód. Po vytvorení signálu oscilátorom je naň napríklad možné aplikovať kompresiu, alebo pripojiť analyzátor a výsledný signál previesť na hash [20]. Vo výsledku sa totiž rovnako spracované signály líšia v závislosti na HW a SW konfigurácie, v článku [20] autori pri testovaní zistili, že aj rôzne verzie toho istého prehliadača na rovnakom zariadení vyprodukovali rôzny odtlačok.

2.2.6 HTTP hlavičky

V HTTP hlavičkách sa štandardne nachádza User-agent string (UAS), ktorý obsahuje informácie o klientovom prehliadači, napríklad typ a verzia prehliadaču, operačného systému,

[3] https://github.com/fingerprintjs/fingerprintjs

Tabuľka 2.2: príklad Audio Context odtlačku

Audio context	channelCount	2
	channelCountMode	explicit
	channelInterpretation	speakers
	maxChannelCount	2
	numberOfInputs	1
	numberOfOutputs	0
	sampleRate	48000
	state	suspended
Analyser node	channelCount	2
	channelCountMode	max
	channelInterpretation	speakers
	fftSize	2048
	frequencyBinCount	1024
	maxDecibels	-30
	minDecibels	-100
	numberOfInputs	1
	numberOfOutputs	1
	smoothingTimeConstant	0.8

preferovaný jazyk a podobne. User-agent sa využíva aby server podľa prezentovaných parametrov vedel klientovi poskytnúť obsah čo najlepšie formátovaný na jeho prehliadač, napríklad *web crawlery* by sa mali prezentovať ako špecifický User-agent. Niektoré prehliadače – napríklad Opera – sú aktualizované tak často, že je verzia veľmi špecifikujúcou informáciou. User agent string môžno podvrhnúť, mnohé populárne rozšírenia to už robia. Pri prepisovaní UAS je dôležité zachovávať konzistentnosť, aby nebolo možné zistiť, že sa jedná o podvrh. Niektoré pluginy vytvárajú náhodné kombinácie parametrov, čo v spojení s inými informáciami získanými z prehliadača niekedy vedie k nereálnym výsledkom [30], alebo k unikátnym UAS.

Vývojári z Googlu už pracujú na zneplatnení UAS, existuje flag na jeho pozastavenie a v budúcnosti by mal byť zastaralý. Tiež odporúčajú vývojárom na ňom nezakladať dôležité prvky stránok, keďže sa jedná o ľahko zneužiteľné údaje a po vypnutí by mohla stránka nesprávne pracovať. Pri viacerých prehliadačoch existujú verejné diskusie na podporu[4]. V budúcnosti by sa mohla ako náhrada používať napríklad Client hints[5].

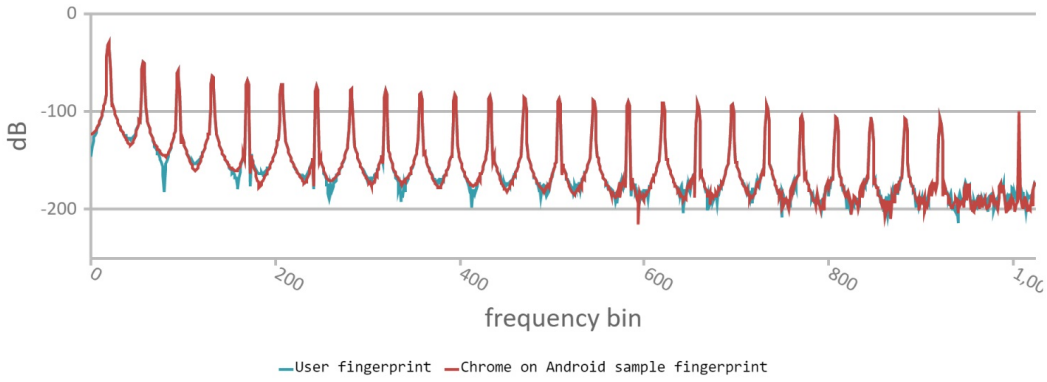
2.2.7 Fonty

Typicky sa zoznamy fontov a pluginov získaval z Flash API, no s príchodom HTML5 sa jeho popularita začala znižovať a momentálne už nie je podporovaný. Ďalšie spôsoby font FP sú napr využitie fontov pri canvas FP, či sidechannel font detection [31]. Pomocou vykresľovania textu rôznymi fontami a merania rozmerov elementu je možné určiť, či je font dostupný a použil sa, alebo sa použil fallback font.

Podobný prístup využíva aj font metrics FP založený na extrakcii rozmerov DOM elementov [22]. Tu cieľom nie je získať zoznam dostupných fontov, ale odtlačok založený na

[4] https://groups.google.com/a/chromium.org/g/blink-dev/c/-2JIRNMWJ7s/m/yHe4tQNLCgAJ

[5] https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/client-hints



Obr. 2.2: Príklad vizualizácie audio signálu v porovnaní so signálom v Chrome pre Android

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.198 Safari/537.36

Obr. 2.3: Príklad UAS

rozdieloch vo vykresľovaní znakov. Pri tejto metóde sa postupne vykresľuje každý znak unikódu rôznymi *font-family* a zapisujú sa rozmery výsledného DOM elementu. Rozdiely nastávajú pri vykresľovaní niektorých lokálnych znakov, či pri iných verziách fontu, tie isté fonty, môžu byť trochu inak vykreslené na rôznych OS, a podobne. Používateľ nevidí, že dochádza k snímaniu odtlačku, no nevýhodou je, že je tento proces časovo náročný a môže trvať aj niekoľko minút. Tento spôsob využitia fontov na snímanie odtlačku je síce slabší, ako niektoré priame techniky - teda fonty sú rozpoznávané s pravdepodobnosťou menšou ako 1, zoznam rozpoznaných fontov nemusí nutne odpovedať reálnemu a počet identifikujúcich bitov je teda menší, pričom celý proces trvá omnoho dlhšie. No vďaka použití nepriamej cesty je tiež ťažšie sa mu brániť a je pravdepodobné, že bude fungovať aj v budúcnosti.

V praxi sa často kombinujú rôzne prístupy font fingerprintingu na získanie čo najpresnejších výsledkov a zároveň sa vyhnúť ochranám, ak niektoré techniky kvôli ochrane nefungujú.

2.2.8 Pluginy

Súčasťou informácií, ktoré možno z JavaScriptu momentálne jednoducho získať je aj zoznam pluginov - zásuvných modulov. Získavanie informácií o pluginoch bolo súčasťou prvých techník snímania odtlačku prehliadaču [30]. Najviac identifikujúcich bitov poskytovali pluginy ako Flash a Silverlight, kedy veľké množstvo verzií znamenalo lepšiu rozlíšiteľnosť. Výhodou používania zoznamu pluginov a informácií o nich je veľmi jednoduchá prístupnosť, nie je potrebné robiť žiadnu analýzu, stačí pomocou getteru získať zoznamy.

Od konca podpory Flashu už nie je reálny dôvod poskytovať v prehliadači informácie o pluginoch, preto napríklad Firefox, aj keď tieto API podporuje, tak vracia prázdne zoznamy.

2.2.9 Senzory

Moderné – najmä prenosné zariadenia, ako smarftóny, tablety, či notebooky, v sebe obsahujú senzory dôležité pre ich funkcionalitu. V posledných rokoch sa prostredníctvom nových JavaScriptových API stali niektoré senzory dostupné aj pre prehliadače.

Ako výskumy [18, 24] ukázali, prehliadače na mobilných zariadeniach majú väčšiu uniformitu spôsobenú HW aj SW obmedzeniami, a je teda obtiažnejšie identifikovať prehliadač podľa odtlačku. Podľa [16] v roku 2018, 3695 z Alexa top 100 000 stránok pristupovalo pomocou skriptov k API senzorov v mobilných zariadeniach. Výskum tiež zachytáva pokusy o snímanie odtlačkov prehliadaču na mobilných zariadeniach, podobne ako na bežných stolových PC. Podľa získaných dát až 63% [16] skriptov, ktoré pristupovali k senzorom vykonávali tiež nejakú formu fingerprintingu (canvas, audio, canvasfont, webrtc, battery). Z toho sa dá usúdiť, že skripty snímajúce odtlačky prehliadaču sa na mobilných zariadeniach snažia kompenzovať vyššiu homogénnosť využitím senzorov.

Battery status API je možné použiť na zistenie stavu nabitia batérie, dobu nabíjania a vybíjania a podobne. Tieto informácie potom možno využiť v odtlačku, napríklad analýzou rýchlosti nabíjania/vybíjania [20]. Vývojári sú si vedomí bezpečnostných rizík, pričom prínos pre web je limitovaný, preto je toto API momentálne označené ako zastarané a neodporúča sa používať, no stále ho niektoré prehliadače podporujú.

Rozhranie pre senzory – *Sensor API* – v sebe zahŕňa niekoľko senzorov, konkrétne: akcelerometer, senzor okolitého svetla, gyroskop, senzor lineárneho zrýchlenia, megnetometer a senzor orientácie. Tieto rozhrania sa dajú využiť na zistenie ich dostupnosti, čo je jednoduchší prístup s väčšou stabilitou. Alebo tiež na vykonávanie meraní, ktoré sa dajú použiť na vytvorenie odtlačku - avšak, keďže ide o veľmi nestabilné javy, je ich možno použiť iba na krátkodobú identifikáciu.

Jedná sa o nové rozhrania, čo je jeden z dôvodov, prečo sa podpora tohto API medzi prehliadačmi líši. Niektoré senzory sú dostupné v experimentálnom móde, ktorý sa musí používateľ zapnúť, iné sú dostupné po povolení od používateľa. Firefox v nových verziách nepodporuje Sensor API a vytvoril si vlastné rozhrania, dostupné iba pre Firefox.

2.2.10 CSS

Okrem toho, že sa dajú informácie získať z HTTP hlavičiek a JavaScriptu. Článok [35] ukazuje techniky, ktorými možno využiť CSS (Cascading Style Sheets) na získanie informácií, ako je napríklad veľkosť okna, alebo podporované jayzky. Postup takéhoto zisťovania informácii funguje tak, že sa využije požiadavok na obsah zo serveru, ako môžeme vidieť vo výpise 2.1. V database.php sa potom spracuje a uloží si reťazec požiadavku a odpovie.

```
1 p{background-image : url("database.php?
2   property=background-image") ; }
```

Výpis 2.1: Príklad CSS s dotazom

Pomocou takéhoto vzorca je vytvorených mnoho pravidiel s požiadavkami, napríklad media-queries podľa rozmerov okna a server potom podľa reťazca požiadavky vie, ktoré sa použili, a môže si tak vytvárať odtlačok prehliadaču. Zaujímavé na tejto technike je, že aj keď nie je možné všetky vlastnosti určiť úplne presne, napríklad pri fontoch, funguje však aj keď je JavaScript úplne vypnutý. Tým pádom je veľmi obtiažne sa proti tomu brániť.

Kapitola 3

Ochrana pred získavaním odtlačku prehliadača

Od roku 2011 kedy vznikli prvé známe browser FP nástroje sa snažia vývojári aj ľudia vo výskume prísť s ochrannými mechaniznami, ktoré by zabránili browser fingerprintingu. Tento problém je omnoho zložitejší, ako by sa mohlo na prvý pohľad zdať. Blokovanie cookies tiež často spôsobuje nefunkčnosť stránok, a to najmä pri prihlasovaní, avšak používateľ ich môže aspoň svojvoľne vymazať. Naproti tomu je takmer nemožné blokovať všetky *data endpointy*, ktoré fingeprinting skripty využívajú a zároveň nespôsobiť nefunkčnosť väčšiny webových stránok. Ďalším problémom je aj neustála premenlivosť prehliadačov, najmä, čo sa týka používaných API, novšie verzie prehliadačov modifikujú niektoré API, alebo dokonca pridávajú nové, čo dáva priestor fingerprinting skriptom využiť nové dáta a to ešte predtým, než majú prehliadače nejakú ochranu. Veľké bezpečnostné riziká sa nachádzajú v pluginoch ako napríklad Flash [1], ktorý bol v posledných rokoch na ústupe a od konca decembra 2020 ho Adobe prestáva podporovať [4]. Koniec podpory bol ohlásený už v roku 2017 a vývojári tak mali dostatok času sa na zmeny pripraviť. Niektoré prehliadače prestali s jeho podporou už skôr, ako bolo nutné, napríklad Safari.

Homogénny odtlačok

Jedným zo spôsobov, ako sa brániť zachytávaniu odtlačku prehliadača je snaha prehliadač maskovať tak, aby bol jeho odtlačok rovnaký ako väčšina iných. Týmto spôsobom pri následnom sledovaní prehliadača nie je možné od seba rozlíšiť konkrétnych používateľov. Tento prísup je pomerne priamočiary a jednoduchý, no pri skutočnom použití prináša niekoľko problémov. Hlavným problémom je isté obmedzenie používateľa – niektoré parametre prehliadača, ako veľkosť okna, dostupné logické jadrá procesoru, musia byť pevne nadefinované a nemenné, aj keď majú používatelia dostupné lepšie prostriedky. Tiež je väčšina API blokovaná, alebo výrazne upravovaná, aby produkovali rovnaký výstup. [30] Ďalším problémom je rozsah dát, z ktorého fingerprinting skripty čerpajú. Aj napriek homogénnemu odtlačku je možné zistiť o používateľovi informácie. Je možné kombinovať fingerprintingu s template útokmi [33]. Template útoky na základe rôznych úloh, ako je napríklad DOM parsing, alebo séria matematických operácií, profilingu a následnej analýzy dokážu rozlíšiť rozdiely aj na základe hardvérových charakteristík ako je napríklad architektúra procesoru, či inštrukčná sada môže viesť k bezpečnostným medzerám, a je možné použiť získané dáta aj na identifikáciu prehliadaču.

Jedinečný odtlačok

Opačný prístup je vytvárať dojem, že sa pri každej relácii prehliadača, či návšteve domény jedná o iný prehliadač. Pre dosiahnutie tohto dojmu sú snímané údaje upravované s využitím pseudonáhodnosti. Čo však prináša niekoľko problémov – pri zamieňaní hodnôt za náhodné (napríklad pomocou spoofing pluginov) môže dochádzať k nekonzistencii odtlačku [30]. Modifikácia hodnôt šumom môže byť pre jednoduché fingerprinting skripty účinná, pretože aj malé odchýlky vedú k rôznym odtlačkom. No sofistikovanejšie prístupy sa pri dostatočnom počte meraní a analýzou môžu aproximáciou priblížiť k reálnym hodnotám a ochrana stráca zmysel [32]. Preto je nutné zvoliť taký prístup, kedy sa v čase merané hodnoty nemenia a ani pri štatistickej analýze nezískame pôvodné hodnoty. V posledom rade môže mať ovplyvňovanie negatívny dopad na používateľský zážitok – napríklad si predstavme náhodné zmeny dimenzií okna prehliadaču, väčšina používateľov by si radšej zvolila komfort okna na celú obrazovku oproti benefitu zvýšeného súkromia. Ďalší problém je tiež, že nie pri všetkých údajoch sa dá jednoducho klamať a podvrhnúť ich, väčšia časť HTTP hlavičky musí ostať rovnaká a pri podvrhnutí user agent stringu môže dôjsť k nesprávnemu zobrazeniu stránky.

Medzi príklady existujúcich riešení sa radia napríklad PriVaricator[32], FPRandom [29], Blink [27], či nová FP ochrana od Brave [14], bližšie popísané v častiach 3.1, 3.2 a 3.3.

3.1 Existujúce riešenia

Táto sekcia obsahuje príklady momentálnych konceptov vo výskume ale aj v praktickom nasadení v moderných prehliadačoch. V populárnych prehliadačoch sú takéto nástroje iba ojedinelé a v prípade ich zahrnutia ich často musí používateľ explicitne zapnúť. I tak však možno pozorovať trend záujmu o implementáciu nejakej formy ochrany pred snímaním odtlačku prehliadača aj v bežných prehliadačoch [19], čo môže naznačovať väčší záujem používateľov o bezpečnosť.

3.1.1 PriVaricator

Hlavnou myšlienkou tohto riešenia je mechanizmus, ktorý podľa zvolenej úrovne vnáša pseudonáhodné hodnoty do reálnych, čím vytvára falošné výstupy. Je možné nastaviť po koľkých prístupoch (ku konkrétnym API endpointom) prehliadač začne „klamať“ a s akou pravdepodobnosťou bude klamať. Tento prístup sa javí ako efektívny z hľadiska početnosti stránok, ktoré používajú skripty na snímanie odtlačku prehliadaču [20]. Problém však nastáva, ak skript nepoužije veľa prístupov a prehliadač nezačne klamať, odtlačok by mohol byť správny. Vývojári tiež nebrali do úvahy testovanie počas dlhšieho intervalu a následnú štatistickú analýzu, na základe veľkého množstva dát by sa dalo dopátrať k reálnym hodnotám.

Po implementačnej stránke bol prehliadač, ktorý vznikol v rámci výskumu len veľmi malou modifikáciou Chromia - finálny patch mal menej ako 1000 riadkov. Cieľom bolo vytvoriť bezpečnejší, no stále rýchly prehliadač. Pri testovaní autori dosahovali veľmi podobné rýchlosti načítavania stránok ako originálny prehliadač, niektoré funkcie boli pomalšie, no ľudsky nerozoznateľným rozdielom.

3.1.2 Blink

Riešenie, ktoré vzniklo v rámci výskumnej skupiny okolo P. Laperdrixa. Jedná sa o demonštračné riešenie, ktoré nie je komerčne dostupné a je založené na spájaní rôznych au-

tentických komponentov – prehliadače, OS, fonty, pluginy a iné na vytvorenie unikátnej inštancie [27]. Blink používa virtualizáciu na spúšťanie rôznych operačných systémov, na ktorých sa spúšťajú spomínané komponenty. Miera ochrany je vysoká, pretože sa jedná o rôzne reálne komponenty, ktoré sa pri spustení kombinujú na virtuálnom stroji, spojenie odtlačkov prehliadačov medzi reláciami je veľmi nepravdepodobné. Jednotlivé komponenty sú autentické a sú ich stovky, môžu zaberať veľa úložného priestoru. Keďže tiež toto riešenie používa virtualizáciu, je náročnejšie na výkon a nie je vhodné pre menej výkonné zariadenia. Dá sa povedať, že Blink je robustné riešenie, ktoré kvôli zvýšenej anonymite vyžaduje viac zdrojov – najmä ukladací priestor.

3.1.3 FPRandom

Laperdrix et al. [29] prezentujú výskum, ktorý zakladá na vnášaní náhodnosti do hodnôt, ktoré sa používajú na získavanie odtlačku prehliadaču. Pracovali na ňom ľudia, ktorí predtým vytvorili Blink, teraz sa zamerali na „lightweight“ prehliadač, ktorý by používateľa príliš neobmedzoval, no stále poskytoval ochranu proti fingerprintingu. Teoretické riešenie je implementované ako upravená verzia prehliadaču Firefox. Pri Canvas API autori modifikujú metódu ParseColor, ktorá sa používa pri zadávaní farieb, zmena spočíva v pridaní náhodného šumu do každej farby, čo používateľ nepostrehne, no pri odtlačku to spôsobí rozdiel. Modifikujú tiež metódu SetFont, používanú pri vyberaní písma a to tak, že sa písma vyberajú náhodne z fontov poskytnutých OS, a pri pokuse o fallback font vyberie font rozdielny oproti minulej relácii. Takto je zaistená unikátnosť Canvas odtlačkov, záleží však aj na počet inštalovaných fontov a spôsobe získavania odtlačku.

AudioContext API – modifkácia metód pracujúcich s AudioBuffer, pridáva sa malý šum, ktorý človek nerozozná, no ovplyvní odtlačok. Zoradzovanie JS objektov – modifikácia správania SortComparatorIds, aby bol spôsob zoradzovania rozdielny medzi reláciami – pri spustení sa náhodne určí jeden zo spôsobov zoradzovania a používa sa do skončenia relácie. FPRandom môže pracovať v dvoch módoch: Random mode – kedy bude výsledný odtlačok iný pri každom meraní, a Session mode – odtlačok bude rovnaký pre jednu reláciu prehliadaču.

Pri testovaní autori dospeli k záveru, že oproti originálnej verzii Firefox nastáva isté spomalenie, no používateľ by si to nemal všimnúť. Riešenie sa tiež ukázalo ako efektívne, keďže vyprodukovalo každé meranie iné odtlačky, či sa však odtlačky nedajú neskôr na základe analýzy spojiť, je otázne. V rámci práce bol urobený aj prieskum medzi malou skupinou používateľov, ktorí boli testovaní na rozpoznanie zmien v obrázku canvasu, či zvuku, ktoré prehliadač modifikoval šumom. Napriek veľmi malým zmenám bola časť testovaných používateľov schopná rozlíšiť zmenené a pôvodné obrázky a zvukové stopy.

3.1.4 Tor browser

Tor browser[1] je známy snahou o anonymizáciu používateľov. Tor prehliadač smeruje komunikáciu automaticky cez Tor sieť, čím skrýva IP adresu klienta a zvyšuje tak anonymitu.

Bol to prvý prehliadač, ktorý implementoval ochranu proti získavaniu odtlačkov a dodnes poskytuje jednu z najlepších ochrán. Jeho ochrana sa zakladá na princípe homogenizácie – všetci používatelia Tor browseru by mali mať rovnaký odtlačok [23]. V praxi to tak však vždy nie je, napríklad preto, že pri zapnutí ochrany proti browser FP sa Tor browser spúšťa v okne s rozlíšením 1000x1000px, ak ho používateľ upraví, tak riskuje unikátnosť

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

odtlačku. Aj keď sa podarí udržať homogénny odtlačok prehliadaču, hrozí riziko, že údaje o OS, či hardvér, krehký odtlačok narušia. Na dosiahnutie homogénneho odtlačku tiež Tor browser blokuje väčšinu JavaScript API, čiže stránky využívajúce canvas, či WebGL, alebo iné API budú nefunkčné alebo budú fungovať obmedzene. Taktiež poskytuje ochranu proti font fingerprintingu pomocou font-whitelistu, ktorý limituje fonty používané v prehliadači.

3.1.5 NoScript

NoScript[2] je jednoduché rozšírenie dostupné na väčšine moderných prehliadačov (a zabudované v Tor browseri), blokujúce spúšťanie JS skriptov. Jedná sa o veľmi efektívnu ochranu voči snímaniu odtlačku prehliadača, bez JavaScriptu zostanú priamo dostupné len informácie z HTTP. Keďže však skoro každá webová stránka používa JavaScript, je ich bežné prehliadanie takmer nemožné. NoScript preto funguje na báze white-listu – na doménach, ktorým používateľ dôveruje, explicitne zapne JavaScript, alebo konkrétne skripty. Toto môže byť niekedy obtiažne, ak predpokladáme, že používateľ neanalyzuje všetky skripty na všetkých stránkach. Z týchto dôvodov je toto rozšírenie populárne skôr medzi nadšencami pre súkromie a bezpečnosť.

3.1.6 Firefox

Firefox od verzie 72 obsahuje novú ochranu súkromia, ktorej súčasťou je aj ochrana pred snímaním odtlačku prehliadaču [19]. V spolupráci so spoločnosťou Disconnect[3], ktorá vytvára produkty na ochranu súkromia. Ochrana proti sníaniu odtlačku vo Firefoxe je odlišná, ako predošlé spomínané prístupy. Namiesto blokovania či upravovania API volaní, ktoré sú používané na získavanie odtlačku, sú blokované požiadavky na servery tretích strán, ktoré vykonávajú cross-domain fingerprinting. Domény vykonávajúce sledovanie na základe odtlačku (a iné škodlivé domény) sú ukladané do zoznamu, ktorého udržiavanie zastrešuje Disconnect[4]. Pridávanie domén do zoznamu je posudzované na podľa meraní, ktoré sú vykonávané na základe predošlých výskumov [16, 20]. Domény vybrané na základe meraní sú ďalej verejne vyhodnocované v rámci Disconnectu. Takýto prístup nijak neobmedzuje používateľa, ale hrozí šanca sledovania pomocou odtlačkov v časovom období dokým je sledujúca doména rozpoznaná a pridaná do blocklistu.

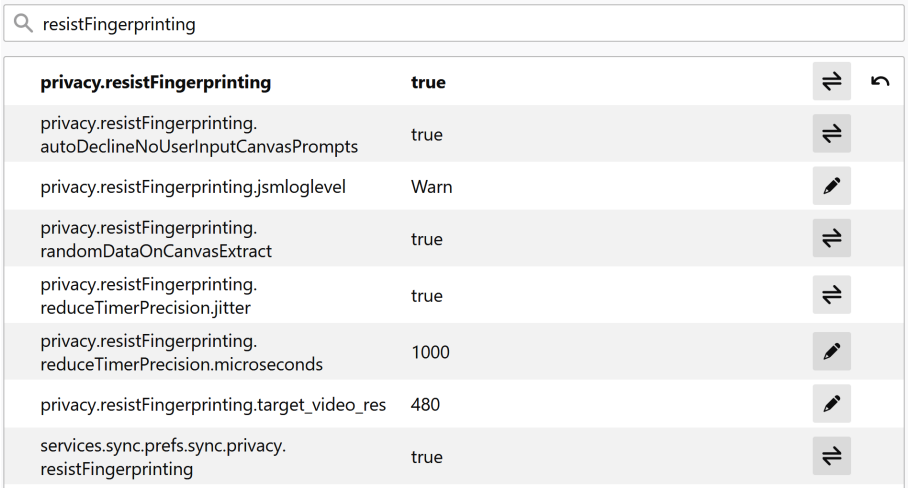
Okrem toho bolo v rámci Tor uplift programu[5] do Firefoxu pridané množstvo funkcionality na ochranu súkromia z Tor browseru. Vďaka tomu Firefox obsahuje aj ochranu proti snímaniu odtlačku, podobnú ako bola spomínaná v časti 3.1.4. Túto funkcionalitu možno zapnúť pomocou konfiguračnej stránky *about:config*, kde je potrebné zapnúť parameter *privacy.resistFingerprinting*, alebo aj iné špecifické parametre, príklad obrazovky s týmito nastaveniami možno vidieť na obrázku 3.1.

[2] https://noscript.net/

[3] https://disconnect.me/

[4] https://github.com/disconnectme/disconnect-tracking-protection/blob/master/descriptions.md

[5] https://wiki.mozilla.org/Security/Tor_Uplift



Obr. 3.1: Výstrižok obrazovky stránky about:config vo Firefoxe

3.2 Javascript Restrictor

JSR (Javascript Restrictor) je rozšírenie pre internetové prehliadače, ktorého účelom je zvýšenie úrovne súkromia a bezpečnosti pri používaní bežných prehliadačov (chromium based, opera, firefox). Toto rozšírenie vzniklo ako diplomová práca pána Červinku [38] pod názvom JavaScript Restricter, pôvodne bolo rozšírenie určené iba pre prehliadač Mozilla Firefox a poskytovalo iba malú funkcionalitu. Odvtedy prešlo úpravou a rozšírením funkcionality v práci [36], ktorá pridala aj prvky ochrany proti snímaniu odtlačku prehliadaču a rozšírenie sa transformovalo na JavaScript Restrictor.

Hlavným zameraním tohto rozšírenia je ochrana proti zneužívaniu medzier vo webovom prehliadači na sledovanie, či zbieranie súkromných údajov používateľov. Do tohto samozrejme spadá aj ochrana proti snímaniu odtlačku prehliadaču, pomocou ktorej je možné sledovať prehliadač naprieč doménami.

Zároveň sa JSR snaží o zachovanie funkčnosti stránok. Napriek zabráneniu nechceného sledovania, či ochrane pred únikmi dát by nemalo rozšírenie obmedzovať funkcionalitu webu.

JavaScript Restrictor ponúka štyri úrovne ochrany, čo sa týka ochrany pred snímaním odtlačku, sú dôležité nasledujúce opatrenia:

Level 0	všetky prvky ochrany sú vypnuté
Level 1	zníženie presnosti času na stotiny sekundy zníženie presnosti performance.now na desiatky zníženie presnosti geolocation API na stovky metrov nahrádzajú sa HW informácie v navigator API blokuje sa battery status API
Level 2	ochrana proti canvas FP zníženie presnosti času na desatiny sekundy zníženie presnosti performance.now na stovky zníženie presnosti geolocation na kilometre
Level 3	blokuje sa geolocation API zníženie presnosti času na sekundy a pridanie náhodnosti zníženie presnosti performance.now na tisícky

Canvas pri zapnutej ochrane sú endpointy *canvas.toDataURL()*, *canvas.toBlob()* a *CanvasRenderingContext2D.getImageData()* zaobalené a modifikované tak, aby vracali prázdny biely obrázok. Toto riešenie chráni pred canvas based font fingerprintingom a niektorými inými špecifickými formami canvas fingerprintingu, no pri populárnych FP skriptoch sa môže ukázať ako neefektívne. Pri pokuse o získanie hodnoty z canvas elementu totiž poskytne vždy rovnakú hodnotu a vzhľadom na malý okruh používateľov JSR by mohlo ísť o identifikujúci údaj.

Navigator Endpointy *navigator.deviceMemory* a *navigator.hardwareConcurrency* môžu odhaľovať hardvérové detaily, preto sa zaobaľujú. *navigator.deviceMemory* sa modifikuje tak, aby vždy vracalo hodnotu 4, *navigator.hardwareConcurrency* hodnotu 2. Obe tieto hodnoty sú totiž často používané. Obaľuje sa tiež metóda *navigator.enumerateDevices*, ktorá vracia media zariadenia (kamery, mikrofóny reproduktory), na zvýšenie anonymity je teda po modifikácii vracaný prázdny zoznam.

Battery status API je rozhranie na sledovanie stavu batérie, pri zapnutej ochrane sa toto rozhranie blokuje, keďže sa dá použiť na vytvorenie krátkodobého odtlačku prehliadaču.

Geolocation Toto rozhranie možno použiť na získanie presnej polohy zariadenia, ak používateľ povolí zdieľanie polohy. Kvôli potvrdeniu sa nedá bežne použiť pri snímaní odtlačku, no v prípade potvrdenia ak ho chce používateľ používať JSR limituje presnosť, alebo ho pri najvyššej ochrane vypne.

Režim ochrany sa aplikuje na konkrétnu doménu, pre ktorú ho používateľ nastaví. Pri nastavení na vybranej úrovne sa doména zapamätá a úroveň sa uloží pre budúce návštevy - jednoducho tak možno znížiť ochranu pre známe a dôveryhodné stránky. Pre všetky domény, pre ktoré používateľ inú úroveň nenastavil sa aplikuje východzia úroveň, ktorá je na začiatku 2 a používateľ ju môže taktiež zmeniť.

Pomocou zníženia presnosti časovačov a pridanie pseudonáhodnosti JSR poskytuje ochranu proti časovaným útokom [37]. Časované útoky fungujú na princípe merania časových úsekov medzi vopred určenými špecifickými úlohami, ako je spracovanie videa, ukladanie a načítavanie cache dát a podobne. Vďaka presnému meraniu je možné odhadnúť špecifikácie

prehliadaču alebo systému, ktorý je podrobený meraniu a toto ďalej použiť pri identifikácii. JavaScript Restrictor teda znepresnením časovačov predchádza podobným meraniam.

V svojej práci, ktorá priniesla väčšinu momentálnej funkcionality [36] pán Timko popisuje, ako účinná je implementovaná ochrana a ako pôsobí proti nástrojom na snímanie odtlačkov. Na konci práce sú zhrnuté získané dáta v grafoch. Z meraní vyplýva, že podvrhovaním canvasu, WebGL parametrov, operačného systému a podobne dochádza k zvýšeniu podobnosti týchto parametrov voči ostatným meraným skupinám. Celková účinnosť je však diskutabilná, keďže ostatné parametre spadali do veľmi málo podobných množín.

Okrem týchto opatrení rozšírenie zahŕňa aj Network Boundary Shield (NBS), ktorý zapúzdruje WebRequest API a zachytáva všetky odchádzajúce požiadavky a zabraňuje tak stránkam, aby používali prehliadač ako proxy server medzi lokálnou sieťou a internetom. NBS je zapnutý vo všetkých úrovniach ochrany a je možné ho vypnúť samostatne.

3.2.1 Fungovanie rozšírenia JSR

JavaScript Restrictor funguje v skratke tak, že sa po zapnutí prehliadaču, či rozšírenia na základe definícii obaľovacích štruktúr vygeneruje obaľovací kód, ktorý sa drží v premennej v skripte na pozadí. Po navštívení stránky sa vygenerovaný kód vloží na začiatok a nahradí tak niektoré metódy, či vlastnosti obaľovacími.

Nižšie môžeme vidieť príklad obaľovacej štruktúry, ktorá sa používa na vytvorenie obaľovacieho kódu a potom sa pridáva do zoznamu wrapperov. Z tohto zoznamu sa neskôr generuje obaľovací kód.

```
1  {
2    parent_object: "Performance.prototype",
3    parent_object_property: "now",
4    wrapped_objects: [
5      {
6        original_name: "Performance.prototype.now",
7        wrapped_name: "origNow",
8      }
9    ],
10   helping_code: rounding_func + noise_func + `
11         let precision = args[0];
12         let doNoise = args[1];`,
13   wrapping_function_args: "",
14   wrapping_function_body: `
15         var originalPerfValue = origNow.call(window.performance);
16         var limit_precision = doNoise ? noise_func : rounding_func;
17         return limit_precision(originalPerfValue, precision)`,
18   }
```

Výpis 3.1: Príklad obaľovacieho kódu v JSR

Na výpise 3.1 môžeme vidieť príklad obaľovacieho kódu, konkrétne ide o obalenie metódy *Performance.now*. Obaľovacia štruktúra má formu JSONu, pričom na začiatku sú nastavené objekty, ktoré sa budú obaľovať, na riadku 7 je nastavené meno pôvodnej metódy, pomocou ktorého ju môžeme volať v tele novej funkcie. Riadok 10 obsahuje pomocný kód, do ktorého sa vkladajú vopred pripravené funkcie, ktoré budeme používať a sú uložené v podobe premennej, alebo je možné vo forme viacriadkového reťazca nové funkcie nadefinovať. Najdôležitejší je riadok 14, na ktorom je v podobe reťazca definované telo novej funkcie, ako vidíme ďalej najprv sa uloží hodnota vrátená pôvodnou funkciou a tá sa ďalej upravuje a nakoniec je vrátená.

3.3 Brave

Prehliadač Brave[6] vznikol v roku 2016, po tom ako Brendan Eich opustil spoločnosť Mozilla a založil Brave software. Prvá stabilná verzia vyšla v roku 2019, kedy bol mesačný počet používateľov 8.7 milióna [10], počet používateľov sa však len ťažko monitoruje nezávislými účastníkmi, keďže sa v HTTP komunikácii prezentuje rovnako ako Chrome. Podľa testov robených spoločnosťou Brave pri vydaní stabilnej verzie bol Brave pri načítavaní stránok niekoľkonásobne rýchlejší ako prehliadač Chrome, pričom využíval znateľne menej pamäte [10]. Relevantnosť takéhoto testovania možno samozrejme spochybniť, keďže nebolo nezávislé, testy prebiehali na stránkach novín a pravidlá merania určovali zamestnanci z Brave software. Brave je momentálne dostupný na platformách Android, Windows, macOS, Linux a iOS.

Tento prehliadač je zaujímavý aj vďaka biznis modelu založenom na BAT(basic attention token)[34]. Poplatky v BAT, ktoré by odmeňovali požívateľov a spoločnosť by mali v Brave nahradiť klasický reklamný model. Tiež by takýto reklamný systém nemal obsahovať sledovače a poskytovať informácie o používateľoch. Okrem toho tento prehliadač zahŕňa úzku integráciu nástrojov a peňaženiek pre kryptomeny.

Brave je tiež známy zameraním na súkromie svojich používateľov. Už od prvých verzií bolo jednou z hlavných funkcií blokovanie reklám a sledovačov. Obsahuje tiež možnosť pri anonymnom režime používať sieť Tor, ktorá by mala poskytovať ešte väčšiu anonymitu, hlavne čo sa týka komunikácie na sieti.

Prehliadač Brave je postavený podobne ako iné prehliadače, napríklad Opera, na základe Chromium[7]. Zdieľa teda Chromium záplaty a vylepšenia, čo je dobré pri rýchlej oprave chýb, ako to býva pri open-source Chromiu zvykom. Je však tiež pravda, že prenositeľnosť nie je úplná, a to kvôli zmenám, ktoré nastali počas vývoja, existuje aj podrobný zoznam zmien, ktorými sa Brave od Chromia fundamentálne odlišuje[8], a ktoré v Brave nie sú podporované.

3.3.1 Brave ochrana pred snímaním odtlačku

Ako bolo uvedené v [10] prehliadač Brave vznikol za účelom ochrany súkromia používateľov. Preto už v prvej stabilnej verzii ponúkal istú ochranu pred snímaním odtlačku prehliadača, niektoré API boli po zapnutí ochrany blokované, čím sa zúžil odtlačok a bolo tak náročnejšie prehliadač rozoznať v pobodných množinách, ako bolo spomenuté v [30]. Účinnosť tohto riešenia bola iba čiastočná, keďže odtlačky mohli byť stále dostatočne identifikujúce, na základe iných vlastností. Najväčším problémom bolo obmedzenie funkcionality stránok pri zapnutí najvyššej ochrany.

Z dôvodu nepostačujúcej ochrany, narastajúci počet stránok, ktoré vykonávali fingerprinting a v snahe ochranu zvýšiť, začali Brave vývojári pracovať na efektívnejších opatreniach [9, 12, 13]. Výsledkom je systém nazvaný Farbling, ktorý vnáša do odtlačkov náhodnosť, čím znemožňuje sledovanie prehliadača naprieč stránkami a reláciami. Farbling vychádza z predošlých prác, najmä Laperdrix et al. [29] a Nikiforakis et al. [32]. Prístup ochrany sa v Brave zmenil z blokovania API na zvyšovanie entropie v odtlačkoch. Brave autori tiež argumentujú tým, množstvo zdrojov informácií, ktoré sa dajú použiť na zber odtlačkov je príliš rozsiahly a stále sa rozširuje, nie je teda reálne stále udržovať konzistentný odtlačok medzi všetkými prehliadačmi a zároveň neohroziť - či priamo vedome nepoškodiť

[6] https://brave.com/

[7] https://www.chromium.org/

[8] https://github.com/brave/brave-browser/wiki/Deviations-from-Chromium-(features-we-disable-or-remove)

používateľský zážitok. Pri každej relácii a eTLD+1[26] sú vytvárané seedy, na základe ktorých sú generované hodnoty, ktoré sa v podobe šumu pripočítavajú k reálnym dátam [14]. Týmto je docielené, že odtlačky na konkrétnych stránkach sú navzájom jedinečné a jedinečné sú aj odtlačky medzi rozdielnymi reláciami. Brave ponúka tri úrovne ochrany proti snímaniu odtlačku.

Off – Toto nastavenie sa odporúča používať pri dôveryhodných stránkach, pretože ochrana je vypnutá, no tiež nehrozí žiadne riziko obmedzenia funkčnosti stránky.

Balanced – Východzie nastavenie, na základe seedov sa vygenerujú hodnoty, ktorými sa upravujú výstupné hodnoty, napríklad pri Canvas API to znamená, že sa mierne pozmenia hodnoty farieb — čo používateľ väčšinou ani nezistí, no vznikne tak unikátny odtlačok. Pri tejto úrovni sú stránky stabilné a riziko znefunkčnenia je malé.

Maximum – Podobne ako pri balanced úrovni sú generované náhodné hodnoty, no reálne hodnoty sa vôbec nepoužijú, vďaka čomu je omnoho ťažšie rozlíšiť prehliadač medzi reláciami. Nahrádzanie reálnych hodnôt náhodnými však môže spôsobiť, že stránky nebudú správne pracovať. Poznámka: v čase vypracovania tejto práce nefungovali všetky funkcie ochrany tak ako bolo popísané/plánované – napríklad bola použitá rovnaká forma ochrany pre Balanced aj pre Maximum úrovne.

3.3.2 Účinnosť

Z teoretického hľadiska je ochrana proti vytváraniu odtlačku prehliadača v Brave nedokonalá a neposkytuje úplnú ochranu. Pri implementovaní správnych sledovacích nástrojov by boli používatelia pravdepodobne ohrození. Problémom je rozsah API, na ktoré sa Farbling používa, ak by sa útočník rozhodol ignorovať tieto API tak by bolo možné prehliadače sledovať aj naprieč reláciami. Taktiež je možné zistiť, že sa nejaká forma randomizácie používa a na základe toho odtlačok odlíšiť od bežných odtlačkov a použiť iné techniky fingerprintingu. V reálnom svete však Brave ponúka jednu z najlepších ochrán pred získavaním odtlačku prehliadača v momentálne dostupných prehliadačoch. Najpopulárnejšie fingerprinting nástroje totiž používajú najmä API, ktoré sa vývojári Farblingom rozhodli pokryť. Ochrana pri skutočnom používaní je vďaka tomu vysoká a používateľský zážitok nie je znehodnocovaný zníženou funkcionalitou webových stránok. Je vhodné tiež poznamenať, že Farbling je stále vo vývoji a možno teda očakávať, že sa paleta API rozšíri a ochrana sa tak ešte zvýši. V čase písania práce boli niektoré API rozpracované a iné naplánované do budúcnosti [14].

3.3.3 Konkrétne API

Stav vývoja momentálnej ochrany pred získavaním odtlačku je možné sledovať na Github projekte[9]. Väčšina API, ktoré boli v rámci projektu naplánované je už aj implementovaná, zbytok sa zrejme presunie do verzie ochrany[10]. Táto časť ďalej popisuje riešenia konkrétnych API.

Canvas

Pri úrovniach ochrany **balanced** a **maximum** sa aplikuje funkcia PerturbPixels. Táto funkcia aplikuje filter, ktorý zmení niektoré málo významné bity pre pseudonáhodne vybrané pixely na základe seedov, ktoré sú získané z kľúčov relácie a domény. Farbling funkcia je

[9] https://github.com/brave/brave-browser/issues/8787
[10] https://github.com/brave/brave-browser/issues/11770

aplikovaná na API endpointy CanvasRendering2dContext.getImageData, HTMLCanvasElement.toDataURL, HTMLCanvasElement.toBlob, OffscreenCanvas.convertToBlob. Aj po otestovaní funguje táto ochrana veľmi dobre a nie je voľným okom rozoznateľné, aké konkrétne pixely boli zmenené.

Web Audio

Pri **balanced** úrovni sa na zvukové dáta aplikuje filter, ktorý na základe seedu, ktorý je získaný z doménového a relačného kľúčca, náhodne upraví hlasitosť audio signálu na náhodných miestach o malé hodnoty (menej, ako1%) Pri **maximum** úrovni sa vygeneruje na základe seedu, ktorý je znovu získaný rovnako, biely šum o malej amplitúde, ktorý nemá nič spoločné s pôvodným streamom. Tým pádom je pri **balanced** úrovni odtlačok unikátny, no má veľmi veľký súvis s pôvodným, kdežto pri úrovni **maximum** je odtlačok úplne jedinečný, aplikácie využívajúce tieto audio funkcie však nebudú správne fungovať.
Filter je podľa úrovne aplikovaný pri API endpointoch:

AnalyserNode.getByteTimeDomainData
AnalyserNode.getFloatTimeDomainData
AnalyserNode.getByteFrequencyData
AnalyserNode.getFloatFrequencyData
AudioBuffer.getChannelData
AudioBuffer.copyFromChannel

enumerateDevices

Táto časť je stále vo vývoji a pravdepodobne sa zmení. Pri úrovniach ochrany **balanced** a **maximum** sa na základe seedu náhodne usporiada poradie vrátených zariadení. Používa sa na endpointe MediaDevices.enumerateDevices.

WebGL

Pri úrovni ochrany **maximum** sa modifkujú vracané hodnoty na API endpointoch na minimálne – endpointy vracajú prázdne objekty, nulu a podobne, podľa typu. Ďalej sa na **maximálnej** úrovni nahrádza pri metóde WebGLRenderingContext.getParameter a argumentoch unmaskedVendor a unmaskedRenderer vracaná hodnota náhodne vygenerovaným reťazcom. Ostatné zahrnuté endpointy:

WebGLRenderingContext.getFramebufferAttachmentParameter
WebGLRenderingContext.getActiveAttrib
WebGLRenderingContext.getActiveUniform
WebGLRenderingContext.getAttribLocation
WebGLRenderingContext.getBufferParameter
WebGLRenderingContext.getExtension
WebGLRenderingContext.getFrameBufferAttachmentParameter
WebGLRenderingContext.getProgramParameter
WebGLRenderingContext.getRenderBufferParameter
WebGLRenderingContext.getShaderParameter
WebGLRenderingContext.getShaderPrecisionFormat
WebGLRenderingContext.getTexParameter
WebGLRenderingContext.getUniformLocation
WebGLRenderingContext.getVertexAttribOffset
WebGLRenderingContext.readPixels

WebGL 2

Platí rovnaká ochrana, ako pri WebGL, ale sú zahrnuté ďalšie endpointy. Pri *WebGLRenderingContext.getParameter* sa okrem toho upravujú návratové hodnoty pri nasledujúcich argumentoch. Keďže sa má pri týchto argumentoch metóda *getParameter* návratový typ Glint, sú pri úrovni **balanced** číselné hodnoty pozmeňované o 1 s istou pravdepodobnosťou.

GL_MAX_VERTEX_UNIFORM_COMPONENTS
GL_MAX_VERTEX_UNIFORM_BLOCKS
GL_MAX_VERTEX_OUTPUT_COMPONENTS
GL_MAX_VARYING_COMPONENTS
GL_MAX_TRANSFORM_FEEDBACK_INTERLEAVED_COMPONENTS
GL_MAX_FRAGMENT_UNIFORM_COMPONENTS
GL_MAX_FRAGMENT_UNIFORM_BLOCKS
GL_MAX_FRAGMENT_INPUT_COMPONENTS
GL_MAX_UNIFORM_BUFFER_BINDINGS
GL_MAX_COMBINED_UNIFORM_BLOCKS
GL_MAX_COMBINED_VERTEX_UNIFORM_COMPONENTS
GL_MAX_COMBINED_FRAGMENT_UNIFORM_COMPONENTS

Na úrovni **maximum** sú hodnoty pri týchto argumentoch tak isto upravované, ak ich nezahŕňa zoznam nižšie. Ďalej sú na úrovni **maximum** modifkované návratové hodnoty pri nasledujúcich argumentoch, a to tak, že sa vracajú spodné hodnoty, teda null, 0 a podobne. Je pravdepodobné, že táto úroveň spôsobí nefunkčnosť väčšiny WebGL aplikácií.

GL_SHADING_LANGUAGE_VERSION
GL_VERSION
GL_COPY_READ_BUFFER_BINDING
GL_COPY_WRITE_BUFFER_BINDING
GL_DRAW_FRAMEBUFFER_BINDING
GL_MAX_VERTEX_UNIFORM_COMPONENTS
GL_MAX_VERTEX_UNIFORM_BLOCKS
GL_MAX_VERTEX_OUTPUT_COMPONENTS
GL_MAX_VARYING_COMPONENTS
GL_MAX_TRANSFORM_FEEDBACK_INTERLEAVED_COMPONENTS
GL_MAX_FRAGMENT_UNIFORM_COMPONENTS
GL_MAX_FRAGMENT_UNIFORM_BLOCKS
GL_MAX_FRAGMENT_INPUT_COMPONENTS
GL_MAX_UNIFORM_BUFFER_BINDINGS
GL_MAX_COMBINED_UNIFORM_BLOCKS
GL_MAX_COMBINED_VERTEX_UNIFORM_COMPONENTS
GL_MAX_COMBINED_FRAGMENT_UNIFORM_COMPONENTS

User Agent

Pri úrovni ochrany **balanced** sa pri desktop verzii vracia upravená hodnota verzie operačného systému. Pri úrovni **maximum** sa na všetkých platformách vracia východzie Chrome zariadenie a na konci user agent string sa pridá na základe seedu niekoľko prázdnych znakov. endpoint: NavigatorID.userAgent

Plugins

Pri úrovni **balanced** sa na základe seedu niektoré slová z nainštalovaných pluginy, ktoré by boli inak vrátené náhodne obmenia slovami z dopredu určeného poľa slov, pridajú sa dva náhodne vygenerované pluginy a poradie vracania pluginov je tiež náhodné.

Pri úrovni **maximum** sa vrátia dva náhodne vytvorené pluginy na základe seedu a poradie je náhodné. endpointy: NavigatorPlugins.plugins, navigator.plugins

Poznámka: Je možné, že táto ochrana sa čoskoro reimplementuje a bude vracať len prázdny zoznam pluginov.

Hardware Concurrency

Pri úrovni **balanced** sa na základe seedu vráti celočíselná náhodná hodnota medzi 2 a skutočnou hodnotou. Pri úrovni **maximum** sa na základe seedu vráti celočíselná náhodná hodnota medzi 2 a 8. endpoint: navigator.hardwareConcurrency

Device Memory

Pri úrovni **balanced** sa na základe seedu vráti valídna náhodná hodnota medzi 0,25 a skutočnou hodnotou. Pri úrovni **maximum** sa na základe seedu vráti valídna náhodná hodnota medzi 0,25 a 8. Valídne hodnoty sú 0,25; 0,5; 1; 2; 4; 8. endpoint: navigator.deviceMemory

3.3.4 Budúci vývoj

Vývoj ochrany pomocou Farblingu sa dostáva do konečnej fázy, kedy väčšina plánovaných API je pokrytá. Aj po dokončení budú samozrejme opravované chyby, prípadne pridávané zlepšenia na základe stavu API. Už teraz však začali vývojári pracovať na ďalšej verzii ochrany, ktorá by mala zahŕňať pokrytie nových rozhraní, alebo tých, pre ktoré nebolo možné efektívne implementovať ochranu vzhľadom na nestálosť a prebiehajúci vývoj, alebo si vývojári neboli istí výslednou kompatibilitou. Nová verzia by mala obsahovať:

WebXR: Úprava návratových hodnôt XRSystem.isSessionSupported, pri úrovni **maximum** vždy bude vždy vracať false. Táto časť bola nakoniec vypustená kvôli zmenám v API, ktoré spôsobili nepotrebnosť ďalších zmien.

enumerateDevices: Pridanie nových prvkov (deviceIDs and labels)

Simplified Timezones: Zahrnutie window.Intl.DateTimeFormat API, plán zjednodušiť/normalizovať vracané hodnoty aby bolo ťažšie použiť ich na fingerprinting.

Fonts: Ochrana pred font fingerprintingom bola plánovaná už od prvej verzie Farblingu, no zmeny, ktoré by to vyžadovalo spôsobili odklad. Pravdepodobne bude princíp znovu založený na seedoch, na základe ktorých bude náhodne generovaný zoznam dostupných fontov. Je možné, že sa pri výbere fontov zohľadnia množiny fontov štandardne dostupných na rôznych platformách, Snyder tiež navrhol štandardné riešenie[11] pre W3C, ktorým by sa vývojári prehliadačov riadili, no diskusia je stále otvorená. Problém implementovania ochrany pred font fingerprintingom spočíva v rôznych spôsoboch akým sa takéto odtlačky získavajú. Ako bolo spomínané v časti 2.2.7, existujú rôzne techniky, závislé na rôznych vlastnostiach a sú aj používané v praxi. Ochrana proti font fingerprintingu by teda musela pokrývať všetky metódy, inak by bola neúčinná.

Rozmery okna: Toto je dlhodobo plánovaná zmena, keďže je to jednoducho prístupný údaj a prehliadač Tor, či Firefox ho dokážu podvrhnúť. Riešenie však nie je jednoduché, keďže Brave nechce meniť reálnu veľkosť, ale klamať o rozmeroch, čo sa dá zase obísť cez techniky využívajúce CSS. Nateraz je teda táto časť odložená.

[11] https://github.com/w3c/csswg-drafts/issues/4497

Kapitola 4

Návrh prenosu bezpečnostných opatrení do JSR

Táto kapitola popisuje návrh prenosu ochrany z Brave do JSR. Najprv je potrebné analyzovať implementáciu ochrany proti snímaniu odtlačku v Brave a porovnať ju s tou momentálne implementovanou v JSR. Po pochopení princípu fungovania ochrany a jej dopadov, je ujasnené, ako by sa mala nová ochrana správať, ako by mali vyzerať jednotlivé úrovne a akým spôsobom bude ochrana, ktorú ideme implementovať, fungovať. Potom je možné prejsť k návrhu procesu prenosu ochrany. Keďže je Brave prehliadač a interne funguje úplne inak, ako rozšírenie implementované v JavaScripte, nebude proces prenosu jednoduchý a priamočiary.

4.1 Analýza Brave kódu

Keďže je prehliadač Brave open source projekt publikovaný pod MPL 2.0 licenciou[1], je jeho zdrojový kód verejne dostupný. Vývojári používajú na zverejňovanie i spoluprácu pri nových zmenách nástroj *git* a službu Github [2].

Jadro prehliadača pre Windows, Linux, macOS a Android má samostatný repozitár[3]. V podstate je celá funkcionalita implementovaná v jazykoch C a C++. Jednou z výnimiek je engine pre blokovanie reklám, ktorý bol v roku 2019 prepísaný do jazyka Rust a zároveň vývojári použili nový algoritmus inšpirovaný rozšírením uBlock[4], čo je jedno z najpopulárnejších a najúčinnejších rozšírení pre blokovanie reklám a sledovačov, pričom nie je náročné na zdroje. Podľa testovaní [11] je novší engine 69 krát rýchlejší ako pôvodný.

Nanešťastie nie je Farbling samostatný modul, ale súhrn štruktúr a funkcií integrovaných do jednotlivých častí prehliadaču. Toto je hlavný dôvod, prečo nemožno použiť nástroje pre analýzu a preklad kódu do JS, ako napríklad Emscripten[5]. Preto bude zdrojový kód prehliadača Brave slúžiť skôr ako inšpirácia, pri snahe implementovať ekvivalentné riešenie zahrnuté do rozšírenia JavaScript Restrictor. Kvôli tomu, že sa pri analýze implementácie ochrany proti snímaniu odtlačku v Brave táto práca často odkazuje na momentálne riešenie a Brave repozitár, je pravdepodobné, že do vydania a aj v budúcnosti sa budú niektoré detaily, či väčšie časti líšiť od aktuálneho stavu v prehliadači Brave.

[1] https://www.mozilla.org/en-US/MPL/2.0/
[2] https://github.com/
[3] https://github.com/brave/brave-core
[4] https://ublock.org/
[5] https://emscripten.org/

4.2 Porovnanie ochrany v Brave a JSR

Ako bolo spomínané v časti 3.2, ochrana proti snímaniu odtlačku prehliadaču v JSR sa snaží prezentovať odtlačok homogénny medzi všetkými prehladačmi. JavaScript Restrictor znižuje šírku odtlačku a to znížením presnosti navracaných hodnôt z Date objektu, performance a geolocation API , blokovaním navigator.battery API, a úpravou window.name, a poskytovaním konštantných hodnôt v API pre canvas, veľkosť operačnej pamäte či počet logických procesorov. Vo výsledku je odtlačok ovplyvnený, výsledky boli aj prezentované v práci [36]. I keď boli zaznamenané zlepšenia, čo sa týka anonymity - čiže zníženie počtu celkových identifikujúcich bitov, skutočná účinnosť a konzistentnosť odtlačku u všetkých zariadení, na ktorú táto ochrana spolieha nebola docielená. Ochrana totiž pokrýva iba malú podmnožinu z pomedzi všetkých API, a na základe celkového odtlačku, ktorý vytvárajú bežné fingerprinting skripty, je možné prehliadač identifikovať, keďže používa dostatok iných endpointov.

Brave na druhú stranu ponúka robustné riešenie, používajúce náhodnosť a pokrývajúce veľké množstvo JavaScriptových API, ako bolo popísané v časti 3.3.3. Brave pokrýva omnoho viac snímaných rozhraní a vývoj stále prebieha, čiže časom budú pridané ochranné prvky aj pre ďalšie. Účinnosť podobných riešení, aj keď nie priamo toho v Brave, popisuje napr. Laperdrix et al. [29] a keďže Brave zahŕňa viac rozhraní s podobným prístupom, ako bolo popisované v publikácii, je dôvod sa domnievať, že táto ochrana je účinná. Účinnosť ochrany v Brave v praxi je tiež demonštrovaná v časti 6.3, kde je s ochranou v prehliadači Brave porovnávané finálne riešenie tejto práce.

4.3 Štruktúra ochrany

Keďže je pri snímaní odtlačku prehliadaču možné využiť obrovské množstvo zdrojov informácií, najmä v rámci JS API, je vhodné upresniť, proti akým útokom bude ochrana slúžiť a aké vlastnosti by mala splňovať. Otázkou teda je, proti akým formám snímania odtlačkov bude nová ochrana smerovaná. Podobne ako Brave sa bude zameriavať na ochranu pred aktívnym snímaním odtlačku prehliadaču a bude používať náhodnosť na produkovanie jedinečných odtlačkov.

Pri implementácii bude teda ochrana rozdelená do rôznych úrovní, podľa účelu.

Úroveň 0 - miernejšia ochrana

Táto úroveň by mala zodpovedať čo najviac úrovni *balanced* v Brave ochrane. Protiopatrenia na tejto stránke by nemali znefunkčňovať stránky, maximálne ich fungovanie trochu ovplyvniť. Budú samozrejme existovať výnimky, kedy niektoré opatrenia ovplyvnia kľúčové fungovanie, ktoré bude znehodnotené[6]. Obaľované metódy by mali produkovať výsledky, ktoré sú pseudonáhodne ovplyvnené, ale stále by mali byť veľmi podobné reálnym výsledkom. Výsledné odtlačky by mali byť rozdielne na rôznych doménach a reláciách. Obalené metódy, či vlastnosti by nemali vracať nevalídne hodnoty, neštandardné valídne hodnoty sú v poriadku, keďže zabezpečujú jedinečnosť odtlačkov. Táto úroveň by sa mala používať bežne, keďže poskytuje najlepší pomer medzi ochranou a používateľským zážitkom.

Úroveň 1 - striktnejšia ochrana

Táto úroveň by sa mala správať rovnako ako úroveň *maximum* v Brave ochrane. Môže teda často dochádzať k znefunkčneniu stránok a to hlavne tých neštandardných, napríklad zakladajúcich svoju funkcionalitu na WebGL API. Odtlačky produkované na tejto úrovni by mali byť rovnako ako na predošlej úrovni jedinečné na rôznych doménach a reláciách, ale výstupné hodnoty by sa už nemali zakladať na reálnych, ak je to možné tak by mali byť náhodné na základe domény a relácie. Ochrana s týmto nastavením je teda vhodná pre špecifické prípady, kedy používateľ požaduje maximálnu ochranu, a to aj za cenu poškodených stránok.

[6] https://pxlsfiddle.com/farbling.html

Úroveň 2 - konzistentný odtlačok

Niektoré rozhrania už boli v JSR obalené, preto bude táto funkcionalita dostupná použitím tejto úrovne. Niektoré časti budú musieť byť upravené, no iné fungujú dostatočne dobre. Táto úroveň by mala produkovať pri pokrytých rozhraniach konštantný odtlačok, avšak pre skutočnú efektivitu by bolo nutné pokračovať vo vývoji a pokryť omnoho viac rozhraní.

Poznámka: Toto je všeobecný návrh opatrení pri konkrétnych rozhraniach, nie celkový návrh úrovní ochrany v JSR, tie budú pravdepodobne rôzne kombinovať rôzne úrovne pri jednotlivých API.

4.3.1 Identifikátory domény a relácie

V Brave je interne pri každom zapnutí generovaný pre prehliadač identifikátor relácie. Pomocou tohto identifikátoru relácie sa pri navštívení stránky generujú a ukladajú do cache identifikátory domény. Na vytváranie identifikátorov sa používa HMAC SHA256, ktorý je inicializovaný s identifikátorom relácie a potom vytvára identifikátor domény. Tým pádom je doménový identifikátor závislý na doméne a relácii.

Pri novej ochrane v JavaScript Restrictore sa budú podobne hashe relácie generovať pri spustení rozšírenia. Pri návšteve webovej stránky sa zase budú generovať a ukladať hashe domény, takže budú jedinečné naprieč reláciami. Fungovanie je síce trochu odlišné, no výsledné správanie by malo zodpovedať tomu v Brave a v prípade potreby je možné túto časť upraviť bez poškodenia fungovania ochrany.

4.4 Prenos

Po analýze zdrojového kódu Brave a pochopení princípu fungovania opatrení pre jednotlivé API bude funkcionalita implementovaná v jazyku JavaScript. Takáto implementácia bude následne začlenená ako rozšírenie funkcionality JavaScript Restrictoru. Niektoré API, ako napríklad *navigator*, či *canvas* sú už zahrnuté v JSR implementácii, tie bude pravdepodobne potreba do istej miery upraviť. Na začiatku práce nebolo jasné, či bude pri všetkých rozhraniach možné vytvoriť ekvivalentne funkčné riešenie ako v Brave, no ako možno vidieť v kapitole 5, všetky zvolené rozhrania boli pokryté ekvivalentne, alebo veľmi podobne ako v Brave.

Ako základ novej ochrany možno použiť momentálnu implementáciu JavaScript Restrictoru. Keďže obsahuje už spomínaný systém obaľovania, do ktorého možno jednoducho pridávať ďalšie časti. Obaľovacie štruktúry budú upravované, alebo budú vytvorené nové, tak ako bolo popísané na konci časti 3.2.

Výsledná ochrana bude podľa výsledkov testovania pravdepodobne zahrnutá ako súčasť už existujúcich úrovní, alebo bude vytvorená nová úroveň ochrany špeciálny pre ochranu inšpirovanú prehliadačom Brave.

Kapitola 5

Implementácia

Táto kapitola popisuje akým spôsobom bola implementovaná ochrana pred snímaním odtlačku prehliadaču do rozšírenia JavaScript Restrictor. Keďže je táto ochrana inšpirovaná tou v Brave, väčšinou sa delí do dvoch úrovní, rovnako ako v Brave, ako bolo spomenuté v časti 3.3.1. Niektoré časti ochrany obsahujú aj tretiu úroveň, ktorá je buď rovnaká, ako v pôvodnej verzii JSR, alebo sa jedná o odlišnú funkcionalitu. Implementácia nadväzuje na predošlé práce [36] a [38].

Systém hashov

Základom ochrany proti zbieraniu odtlačku prehliadaču, ktorá bola v rámci tejto práce implementovaná, sú jedinečné identifikátory konkrétnej relácie a domény vo forme hashu, ako bolo spomínané v návrhu. Hash pre reláciu sa vygeneruje pri spustení rozšírenia, generácia náhodného identifikátora je zaistená pomocou API *Crypto.getRandomValues()*, ktorá zaručuje kryptograficky silnú náhodnosť, a je uložený pomocou *chrome.storage*[1] API pod kľúčom *sessionHash*. Podobne je generovaný a ukladaný aj hash, a to pri načítaní novej domény, tiež je ukladaný pomocou *chrome.storage* vo formáte *doménové meno : hash domény*, tým pádom je doménový hash unikátny aj pre reláciu, aj pre doménu.

Na ukladanie hashov bolo najprv plánované použiť API sessionStorage[2], no počas vývoja sa vyskytol problém so synchronizáciou naprieč kartami, keďže sessionStorage je platná iba v konkrétnej karte, bolo by nutné použiť localStorage[3] na synchronizáciu kariet a záznamy v localStorage, kde sa záznamy zachovajú aj po zatvorení prehliadaču, potom manuálne odstraňovať. Práve pre tieto nedostatky bola zvolená alternatíva storage.storageArea[4].V prípade storage.storageArea je platnosť záznamov do konca relácie, možno tak narozdiel od sessionStorage priamo pristupovať k rovnakým záznamom naprieč kartami a nie je nutná manuálna synchronizácia. Ukladanie a načítavanie je vykonávané hromadne a asynchrónne , vďaka čomu je rýchlejšie ako pri localStorage.

Doménové identifikátory sú generované pre každú novú doménu, čo sa líši od pôvodného prístupu v Brave, kde sa používa eTLD+1. Problémom však je, že neexistuje jednoznačné pravidlo, ktoré by determinovalo efektívne úrovne domén. Bolo by možné použiť Public Suffix List[5], alebo balíčky, ktoré ho v sebe zahŕňajú[6]. Problém s takýmto riešením však môže byť zbytočná veľkosť a ďalšie spomalenie.

Pôvodná implementácia tohto rozšírenia fungovala tak, že sa pri zapnutí rozšírenia (väčšinou pri spustení prehliadaču) sa vygenerovali wrappery (obaľovací kód), tento kód bol na pozadí uložený do premennej a potom sa vkladal do stránok až pri načítaní. Problém tohto prístupu však bol v tom, že nebolo možné pridať dynamické premenné, ktoré boli potrebné pri implementovaní

[1] https://developer.chrome.com/docs/extensions/reference/storage/
[2] https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage
[3] https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
[4] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/storage/StorageArea
[5] https://publicsuffix.org/list/
[6] https://github.com/gorhill/publicsuffixlist.js

novej ochrany. Z tohto dôvodu bol upravený skript *document_start.js*, v ktorom sa volá funkcia na vkladanie vygenerovaného kódu do stránky. Je tu pridaná časť, ktorá získava z *chrome.storage* uložené relačné a doménové hashe, tie sa potom spolu so zdrojovým kódom generátoru čísiel *Alea.js* pripoja v anonymnej funkcii k vygenerovanému kódu a spolu sú vkladané do stránok. Jedná sa teda o mierne pozmenený pôvodný prístup, ale stále tu môže dochádzať k problémom a to hlavne pri zväčšovaní objemu obaľovacieho kódu – ak sa vloženie kódu nevykoná dostatočne rýchlo, môže si stránka napríklad uložiť pôvodné funkcie a potom ich používať, alebo sa môže stať, že sa vykonajú niektoré fingerprinting funkcie skôr ako sú endpointy obalené. Tento problém môže nastať iba pri asynchrónnych volaniach.

Generovanie náhodných čísiel

Všetky časti ochrany používajú nejakú formu pseudonáhodnosti, je však dôležité, aby boli odtlačky na jednej doméne a relácii konzistentné – čiže po obnovení karty by mal odtlačok zostať rovnaký. Na to je nutné, aby postupnosti náhodne generovaných čísiel boli po obnovení rovnaké. Takéto správanie vyžaduje deterministický generátor náhodných čísiel a keďže JavaScript funkcia *Math.random()* nepodporuje seed, bola možnosť generátor buď implementovať, alebo použiť knižnicu. Najmä kvôli rýchlosti bola uprednostnená knižnica a to konkrétne *Alea.js*[7]. Jedná sa o open source implementáciu pôvodného Alea algoritmu, ktorý navrhol v roku 2010 Johannes Baagøe, vydanú pod licenciou MIT[8]. Podľa [5] je tento algoritmus dokonca rýchlejší ako natívna funkcia *Math.random()*, pričom má pre naše účely prijateľnú periódu opakovania $\sim 2^{116}$. V tejto implementácii je možné použiť tri metódy - *quick* (alebo default metóda) - ktorá vracia 32bitový float z rozsahu (0,1), *double* - ktorá vracia 53bitový float z rozsahu (0,1), *int32* - ktorá vracia 32bitové celé číslo. Generátor sa pri načítaní stránky inicializuje s hashom domény, aby boli generované hodnoty pre každú doménu a reláciu unikátne. Zvažované bolo aj inicializovanie s hashom, ktorý by vznikol ako produkt operácie (napríklad xor) nad hashom relácie a domény, no vzhľadom na unikátnosť doménových hashov, ako bolo spomínané v 5 to v momentálnej implementácii nie je potrebné.

Okrem toho sa v niektorých častiach používa na generovanie čísiel a postupností čísiel posuvný register s lineárnou spätnou väzbou. Implementované sú rovnako ako v Brave. Keďže sa v origináli pracuje so 64 bitovými celými číslami a v JavaScripte sú celé čísla kvôli implementácii presné iba do 2^{53}. Jedna verzia teda pre 64 bitové čísla, využíva špeciálny objekt BigInt[9], pomocou ktorého je možné reprezentovať veľmi veľké čísla. BigInt však podporuje operácia iba medzi rovnakými typmi, preto je nutné čísla pretypovávať pomocou volania *BigInt()* konštruktoru, a naspäť potom konštruktorom *Number()*. Na verziu pre 32 bitové čísla stačia primitívne čísla.

5.1 Canvas

Ako bolo spomínané v 3.3.1, Brave v oboch úrovniach ochrany proti snímaniu odtlačkov prehliadaču chráni canvas elementy rovnakým spôsobom, teda mierne upravuje hodnoty niektorých pixelov, čo nenaruší činnosť stránok a používateľ by si zmeny v prípade prepísania canvasu novým obrázkom nemal všimnúť. V tejto práci bola reimplementovaná aj ochrana, ktorá pri príslušných metódach vracia biely obrázok, čo nadväzovalo na pôvodné riešenie v JSR. Pôvodné riešenie obsahovalo obaľovací kód pre canvas metódy, je teda možné využiť stávajúcu štruktúru, do ktorej stačí pridať časti pre nové metódy. Telá pôvodne obaľovaných metód budú upravené do podoby, ktorá bude vyhovovať navrhovanej ochrane a teda produkovať vždy unikátny odtlačok, správanie by malo byť podobné ochrane v Brave.

Táto ochrana obsahuje dve úrovne nastavenia, príslušné API sú obaľované nasledovne:

Úroveň 0 - miernejšia ochrana: Na tejto úrovni sa dáta získavané z canvasu mierne pozmenia, čo by však ani po následnom vykreslení nemalo byť pozorovateľné. Vďaka tomu, že sú dáta

[7] https://github.com/davidbau/seedrandom/blob/released/lib/alea.js
[8] https://opensource.org/licenses/MIT
[9] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigInt

pozmeňované v závislosti na hashi domény a relácie sú potom canvas odtlačky jedinečné na rôznych doménach a naprieč reláciami.

Okrem toho bola pridaná aj ochrana funkcií na testovanie polohy bodov, pričom narozdiel od Brave, kde tieto metódy vracajú vždy *false*, táto implementácia vykonáva zmeny výsledkov na *false* s pravdepodobnosťou 5%. Nakoľko je takýto prístup účinný proti snímaniu odtlačku, je ťažko vyhodnotiteľné.

- ***CanvasRenderingContext2D.getImageData***: V obaľovacom kóde tejto metódy sa vytvorí nový dočasný canvas element s rozmermi originálneho. Do tohto falošného canvasu sa skopíruje obsah pôvodného a na tieto dáta je potom aplikovaný filter, ktorý na základe hashu domény a relácie prechádza vybrané položky, vyberané na základe posuvného registra s lineárnou spätnou väzbou, a mierne upravuje hodnoty. To sa prejaví na zmene vo farbách malého množstva pixelov, vďaka čomu je získaný obrázok vždy unikátny. Vďaka tomu, že sa pracuje s falošným elementom a upravujú sa jeho dáta, pôvodný canvas ostáva nezmenený. Nakoniec sa zavolá nad falošným canvasom originálna metóda, ktorá vráti získané dáta.
- ***HTMLCanvasElement.toDataURL***: Obaľovací kód vytvorí nový canvas element o veľkosti pôvodného, nad pôvodným canvas kontextom sa zavolá *getImageData*, ktorá vráti už modifikované dáta. Tieto dáta sa vložia do falošného canvasu a na ten sa zavolá originálna metóda *toDataURL*. Tým pádom zostalo pôvodné plátno zase nezmenené.
- ***HTMLCanvasElement.toBlob***, ***OffscreenCanvas.convertToBlob***: Obaľovanie funguje rovnako, ako pri predošlej metóde *getImageData*.
- ***CanvasRenderingContext2D.isPointInStroke*** a ***CanvasRenderingContext2D.isPointInPath***: v obaľovacom kóde týchto API sú volané pôvodné funkcie, následne je pravdepodobnosťou s 5% výsledok zmenený na false. Takýto prístup zaručuje zaručuje unikátne výsledky testovania polohy bodov, môže však spôsobiť nekonzistentnosť pri rozsiahlejšom testovaní bodov v statických tvaroch, a viacnásobné testovanie rovnakého bodu môže vracať rôzne výsledky. Tieto problémy by však nemali spôsobovať nefunkčnosť stránok a pri bežnom používaní by si to nemal používateľ všimnúť.

Úroveň 1 - konzistentný odtlačok: V tejto úrovni metódy na získavanie dát z canvasu vracajú biely obrázok, jedná sa teda o striktnejšiu ochranu, ktorá istým spôsobom obmedzuje používateľský zážitok znefunkčnením niektorých stránok používajúcich tieto operácie v náväznosti. Tým, že sa vracia prázdny biely obrázok, by mal byť odtlačok konštantný, čo sa dá použiť v kombinácii s ostatnými konštantnými prvkami, no hrozí riziko, že skript rozpozná, že sa jedná o uniformné spodné hodnoty, aj keď nie je jasné, či skripty na snímanie odtlačkov canvasu takúto funkcionalitu momentálne obsahujú. Táto úroveň sa neodporúča používať v kombinácii s novými prvkami ochrany, ktoré poskytujú unikátny odtlačok. Pôvodná implementácia v predošlej verzii JavaScript Restrictoru obsahovala podobnú ochranu, no nefungovala úplne správne a nepokrývala všetky metódy. Ako aj môžeme vidieť v práci [36], alebo v starších commitoch hlavného repozitáru, pôvodné obalenie pri metóde *toDataURL* sa nahradilo pôvodný canvas bielym obrázkom, podobne pri *getImageData* bol pôvodný canvas vymazaný. V skratke to znamenalo, že po zavolaní týchto funkcií bol ovplyvnený pôvodný canvas, s ktorým na stránke pracujeme, bol nahradený bielym obrázkom, čo značne obmedzuje používateľa pri webových aplikáciách pracujúcich s týmto API . Tieto nedostatky boli v tejto práci odstránené a obaľovanie funguje nasledovne:

- ***CanvasRenderingContext2D.getImageData***: V obaľovacom kóde tejto metódy sa vytvorí nový canvas element s rozmermi toho pôvodného a je vyplnený bielou farbou.
- ***HTMLCanvasElement.toDataURL***: Obaľovanie funguje rovnako ako v úrovni 0, až na to, že falošný canvas je biely.
- ***HTMLCanvasElement.toBlob***, ***OffscreenCanvas.convertToBlob***: Obaľovanie funguje rovnako, ako pri predošlej metóde *getImageData*.
- ***CanvasRenderingContext2D.isPointInStroke*** a ***CanvasRenderingContext2D.isPointInPath***: Obalené funkcie volajú pôvodné funkcie a vracajú reálne výsledky a to z dôvodu,

že ostatné canvas metódy na tejto úrovni produkujú konzistentné odtlačky. Počas vývoja sa testovala aj verzia, kedy by tieto funkcie vracali vždy *false*, rovnako ako to robí Brave, no kvôli obmedzení funkcionality a možnému znefunkčneniu stránok sa od toho upustilo.

5.2 Audio

Táto ochrana zahŕňa API AnalyserNode a AudioBuffer a niektoré metódy, pomocou ktorých sa z nich dajú získať informácie, ktoré sú potenciálne použiteľné na vyhotovenie odtlačku. Vychádza z ochrany v Brave, kde je zvukový signál mierne upravený, alebo nahradený bielym šumom. Nahradzovanie bielym šumom je striktnejšia ochrana, ktorá neprezrádza informácie o pôvodnom signále, no hrozí tu riziko jednoduchej detekcie bieleho šumu a ignorovania tejto časti odtlačku.

API sú obaľované nasledovne:

Úroveň 0 - miernejšia ochrana: Táto úroveň mierne pozmeňuje amplitúdu signálov pri obalených metódach. Malé zmeny závislé na doménovom hashi budú produkovať unikátny odtlačok, ale nespôsobujú nefunkčnosť stránok využívajúcich tieto API, prinajhoršom nepatrné odlišnosti v zvuku. Metódy sú obaľované nasledovne:

- ***AudioBuffer.getChannelData***: V obaľovacom kóde sa zavolá originálna metóda, ktorá vracia typované pole s dátami. Na toto pole sa aplikuje filter, ktorý na základe hashu domény mierne upraví hodnoty poľa, a výsledné pole vráti. Vďaka tomu sa vytvorí jedinečný výstup na každej doméne a relácii.
- ***AnalyserNode.getByteTimeDomainData***, ***AnalyserNode.getFloatTimeDomainData***, ***AnalyserNode.getByteFrequencyData***, ***AnalyserNode.getFloatFrequencyData***,***AudioBuffer.copyFromChannel***: Pri týchto metódach obaľovací kód zavolá pôvodnú metódu, v ktorej je jeden z argumentov pole, do ktorého sa vracia výsledok. Na toto pole sa aplikuje rovnaký filter ako pri *getChannelData*.

Úroveň 1 - striktnejšia ochrana: V tejto úrovni sú striktnejšie obmedzenia a nižšie spomínané metódy produkujú namiesto očakávaných výstupov biely šum závislý na hashe domény, vďaka čomu je odtlačok špecifický pre doménu a reláciu. Metódy sú obaľované nasledovne:

- ***AudioBuffer.getChannelData***: V obaľovacom kóde sa zavolá originálna metóda, ktorá vracia typované pole s dátami, ktoré sa v tomto prípade nepoužijú, potrebujeme iba typ a dĺžku poľa. Obsah poľa sa prepíše bielym šumom (hodnoty z rozsahu 0 až 0.1), ktorý je generovaný posuvným registerom s lineárnou spätnou väzbou.
- ***AnalyserNode.getByteTimeDomainData***, ***AnalyserNode.getFloatTimeDomainData***, ***AnalyserNode.getByteFrequencyData***, ***AnalyserNode.getFloatFrequencyData***,***AudioBuffer.copyFromChannel***: Pri týchto metódach obaľovací kód zavolá pôvodné metódu, v ktorej je jeden z argumentov pole, do ktorého sa vracia výsledok. Obsah poľa sa prepíše bielym šumom, rovnako ako pri *getChannelData*.

5.3 WebGL

Keďže je WebGL rozsiahle API, táto časť je ešte ďalej rozdelená na canvas ochranu a ochranu WebGL vlastností. Celková ochrana má dve úrovne nastavenia - Úroveň 0 znáhodňuje canvas a niektoré WebGL endpointy, nemala by spôsobovať tak nefunkčnosť stránok, úroveň 1 vracia pri canvas metódach biely obrázok a pri niektorých WebGL endpointoch sú vracané spodné hodnoty. Správanie v týchto úrovniach je bližšie popísané v príslušných častiach. Keďže momentálne WebGL API obsahuje dve štandardné verzie a nie každý prehliadač podporuje novšiu verziu. Preto sú obaľovacie funkcie aplikované na *WebGLRenderingContext* aj *WebGL2RenderingContext*, čo kvôli momentálnej implementácii obaľovania spôsobuje redundanciu kódu, čo je však možné v budúcnosti zlepšiť.

5.3.1 Canvas

Podobne ako bolo spomínané v predošlej časti 5.1 o ochrane canvasu, API *WebGLRenderingContext* (a aj WebGL2) tiež využíva canvas narozdiel od 2d obrázkov sa v ňom vykresľujú 3D scény. *WebGLRenderingContext* tiež obsahuje funkciu *toDataURL*, avšak neobsahuje *getImageData*, ktorá bola pri 2D canvase používaná na aplikovanie filtru. Na aplikovanie identických zmien ako pri 2D canvase je obsah WebGL canvasu skopírovaný do nového dočasného 2D canvasu a funkcia je *toDataURL* je zavolaná nad ním. Takéto riešenie síce zdanlivo zbytočne volá funkciu znovu, no na to, aby mohli byť dáta plátna upravené je nutné sa k nim dostať, čo je pri WebGL kontexte možné iba funkciou *readPixels*, ktorá je niekoľkonásobne pomalšia, nevracia dáta v správnom formáte a sú načítavané opačne (od konca). Tým pádom je rýchlejšie zavolať funkciu znovu s dočasným 2D canvasom a delgovať výsledok.

Obalená je aj metóda *readPixels*. Obaľovací kód zavolá originálnu funkciu a na výsledné dáta aplikuje filter rovnako ako pri 2D canvase, ale keďže sú dáta načítavané opačne (v podobe invertovanej matice) nastáva rozdiel oproti 2D canvasu, kde metódy načítavajú dáta v bežnom smere. To, že je filter v podstate invertovaný, keďže sú invertované dáta, však nepredstavuje problém, pretože tieto dáta aj tak nie sú úplne totožné a aj keby bol výsledok *readPixels* invertovaný a znovu zobrazený, tak by používateľ nespozoroval zmeny medzi filtrami, ale skôr zmeny kvôli rozdielnemu formátu, či spôsobu načítavania, príklad možno vidieť na obrázku 5.1.

Obr. 5.1: Porovnanie výsledku invertovaných dát získaných metódou readPixels(naľavo) a dát získaných metódou toDataURL (napravo).

Ochrana API na čítanie canvasu pre *WebGLRenderingContext*, teda pre funkcie *toDataURL* a *readPixels*, funguje v dvoch úrovniach. Na Úrovni 0 je na dáta aplikovaný rovnaký filter, ako na 2D canvas. Na Úrovni 1 *toDataURL* vracia dáta bieleho obrázku o veľkosti plátna, rovnako ako pri 2D canvase, no *readPixels* neupravuje vstupné pole a necháva ho prázdne.

5.3.2 WebGL vlastnosti a metódy

Práca s WebGL API vyžaduje nastavovanie rôznych parametrov a prístup k nim. Keďže sa tieto vlastnosti dajú zneužiť na získanie odtlačku, ako bolo spomínané v časti 2.2.4, bola implementovaná ochrana založená na riešení Brave[10].

getParameter

Táto metóda je sama o sebe dosť komplexný, preto je popísaná samostatne. Prakticky funguje tak, že pri zavolaní s enumom parametru ako argument príslušný parameter vráti (prípadne chybovú hodnotu).

Úroveň 0 - miernejšia ochrana:

Nasledujúca tabuľka obsahuje zoznam vstupných argumentoch, pre ktoré je metóda *getParameter*

[10] https://github.com/brave/brave-core/tree/master/chromium_src/third_party/blink/renderer/modules/webgl

na tejto úrovni v obaľovacom kóde modifikovaná. Návratová hodnota pri týchto parametroch je typu GLint, čiže celé číslo. V obaľovacom kóde je volaná originálna funkcia a výsledky (v prípade, že nie sú 0), sú s 50% pravdepodobnosťou znížené o 1, táto hodnota je potom vrátená. Tento prístup sa spolieha, že fingerprinting skript nebude zisťovať parametre opakovane, v takom prípade by mohol zistiť, že dostáva pre rovnaké parametre iné hodnoty, čo by mohlo prezradiť, že sú hodnoty upravované. Táto nedokonalosť by sa dala ošetriť ukladaním všetkých parametrov do *chrome.storage* podľa doménového mena.

MAX_VERTEX_UNIFORM_COMPONENTS
MAX_VERTEX_UNIFORM_BLOCKS
MAX_VERTEX_OUTPUT_COMPONENTS
MAX_VARYING_COMPONENTS
MAX_TRANSFORM_FEEDBACK_INTERLEAVED_COMPONENTS
MAX_FRAGMENT_UNIFORM_COMPONENTS
MAX_FRAGMENT_UNIFORM_BLOCKS
MAX_FRAGMENT_INPUT_COMPONENTS
MAX_UNIFORM_BUFFER_BINDINGS
MAX_COMBINED_UNIFORM_BLOCKS
MAX_COMBINED_VERTEX_UNIFORM_COMPONENTS
MAX_COMBINED_FRAGMENT_UNIFORM_COMPONENTS
MAX_VERTEX_ATTRIBS
MAX_VERTEX_UNIFORM_VECTORS
MAX_VERTEX_TEXTURE_IMAGE_UNITS
MAX_TEXTURE_SIZE
MAX_CUBE_MAP_TEXTURE_SIZE
MAX_3D_TEXTURE_SIZE
MAX_ARRAY_TEXTURE_LAYERS

Tabuľka 5.1: Argumenty, pri ktorých *getParameter* vracia upravené celočíselné hodnoty

Pre argumenty UNMASKED_VENDOR a UNMASKED_RENDERER sa originálna funkcia nevolá, vygenerujú sa náhodné reťazce obsahujúce alfanumerické znaky a tri špeciálne znaky −_␣ o dĺžke 8, a tieto reťazce sú vrátené na výstup. Dôležité je, že sú tieto reťazce konštantné a nemenia sa pri opakovanom volaní.

Pri volaní metódy *getParameter* s ostatnými vstupnými argumentmi sa zavolá originálna funkcia a výstup sa deleguje.

Úroveň 1 - striktnejšia ochrana: Keďže je táto úroveň ochrany prísnejšia, návratové hodnoty nie sú modifikované o malú hodnotu, ale sú vracané minimálne hraničné hodnoty, čiže prázdny reťazec, null a podobne.

Nasledujúca tabuľka zhŕňa argumenty, pri ktorých má metóda *getParameter* vracia typ GLint, preto obaľovacia funkcia vracia 0.

Nasledujúca tabuľka zhŕňa argumenty, pri ktorých má metóda *getParameter* ako návratovú hodnotu reťazec, preto obaľovacia funkcia vracia prázdny reťazec.

Pri argumentoch v nasledujúcej tabuľke je pri metóde getParamater je vykonaný obaľovací kód, ktorý vráti *null*. Pôvodne majú byť pri týchto argumentoch vracané rôzne objekty, alebo *null* v prípade chyby, čiže by mala byť dodržaná kompatibilita stránok, v prípade, že sú napísané korektne. Pre argumenty UNMASKED_VENDOR a UNMASKED_RENDERER je návratová hodnota náhodný reťazec, rovnako ako v úrovni 0.

Pri volaní metódy *getParameter* s ostatnými vstupnými argumentmi sa zavolá originálna funkcia a výstup sa deleguje.

Ostatné endpointy

Ďalej sú obaľované niektoré endpointy, ktoré môžu byť použité na fingerprinting. Boli zvolené metódy, ktoré modifikuje Brave a niektoré ďalšie, ktoré boli pri testovaní spozorované u fingerprinting

MAX_VERTEX_UNIFORM_COMPONENTS
MAX_VERTEX_UNIFORM_BLOCKS
MAX_VERTEX_OUTPUT_COMPONENTS
MAX_VARYING_COMPONENTS
MAX_TRANSFORM_FEEDBACK_INTERLEAVED_COMPONENTS
MAX_FRAGMENT_UNIFORM_COMPONENTS
MAX_FRAGMENT_UNIFORM_BLOCKS
MAX_FRAGMENT_INPUT_COMPONENTS
MAX_UNIFORM_BUFFER_BINDINGS
MAX_COMBINED_UNIFORM_BLOCKS
MAX_COMBINED_VERTEX_UNIFORM_COMPONENTS
MAX_COMBINED_FRAGMENT_UNIFORM_COMPONENTS

Tabuľka 5.2: Argumenty, pri ktorých *getParameter* vracia 0

VERSION
VENDOR
RENDERER
SHADING_LANGUAGE_VERSION

Tabuľka 5.3: Argumenty, pri ktorých *getParameter* vracia reťazce prázdne

COPY_READ_BUFFER
COPY_WRITE_BUFFER
COPY_READ_BUFFER_BINDING
COPY_WRITE_BUFFER_BINDING
FRAMEBUFFER_BINDING

Tabuľka 5.4: Argumenty, pri ktorých *getParameter* vracia objekty

skriptov a zároveň by nemali spôsobiť ďalšiu nefunkčnosť stránok, príklad môžu byť endpointy getExtension, getSupportedExtensions, ktoré sa nesprávali v Brave podľa očakávania, preto bol aj vytvorený issue[11] v Brave repozitári, ktorý poukazuje na problémy a pravdepodobne sa budú riešiť. Implementácia ochrany v JSR nasleduje špecifikácie.
Úroveň 0 - miernejšia ochrana: Na tejto úrovni by nemali byť stránky znefunkčnené, niektoré z týchto vlastností sú kľúčové k fungovaniu scény a preto nie sú upravované. Vnášanie náhodnosti do väčšiny týchto návratových hodnôt by ani nebolo prakticky možné, keďže časť z nich sú objekty, a pri číselných typoch by to mohlo spôsobovať porušenie scény. Preto obaľovací kód pri všetkých ostatných metódach volá pôvodné metódy a deleguje návratovú hodnotu.

Úroveň 1 - striktnejšia ochrana: Nasledujúce metódy sú obalené, pričom obaľovací kód nevolá pôvodné metódy, ale vytvára a vracia spodné hodnoty. Obaľujú sa len nižšie uvedené metódy, ostatné ostávajú nezmenené.

- **getFramebufferAttachmentParameter**, **getActiveAttrib** , **getBufferParameter**, **getProgramParameter**, **getRenderbufferParameter**, **getShaderParameter**: pri týchto metódach sa v obaľovacom kóde volajú funkcie vytvorené na získanie spodnej hodnoty (podobne ako to bolo pri metóde *getParameter*) podľa vstupných argumentov, tieto spodné hodnoty sú 0, prázdny reťazec a podobne.

[11] https://github.com/brave/brave-browser/issues/15882

- **getActiveUniform**: obaľovací kód vytvorí nový objekt *WebGLActiveInfo*[12] s nulovými vlastnosťami a vráti tento objekt.
- **getAttribLocation**: obaľovací kód vráti *-1*
- **getVertexAttribOffset**: obaľovací kód vráti *0*
- **getShaderPrecisionFormat**: obaľovací kód vytvorí nový objekt typu *WebGLShaderPrecisionFormat*[13] s nulovými vlastnosťami a následne tento objekt vráti.
- **getTexParameter**, **getUniformLocation**, **getExtension**: obaľovací kód pri týchto metódach vracia *null*
- **getSupportedExtensions**: obaľovací kód vytorí a vráti prázdne pole

5.4 Plugins

Počas spracovania tejto práce dochádzalo k zmenám v rendering engine Blink, ktorý používa prehliadač Chrome. Od verzie Chrome 90.0.4430 malo dôjsť k zmene API *navigator.plugins* a *navigator.mimeTypes*, ktoré mali vracať prázdne zoznamy [2]. Pri testovaní v Chromiu sa však prejavili problémy s kompatibilitou niektorých stránok. Nakoniec táto zmena nebola súčasťou vydanej verzie Chrome, no napriek tomu sa dá očakávať, že bude zakomponovaná do niektorej z budúcich verzií prehliadaču, keďže rovnaká funkcionalita je už súčasťou Firefoxu a od konca podpory Flashu (viď časť 3) už nie je potrebná. Je teda možné, že časť tejto ochrany bude v budúcnosti zbytočná, no napriek tomu bola implementovaná ochrana zakladajúca sa na modifikácii týchto API v Brave. Proces zmien v Brave možno sledovať v príšlušnom github issue[14].

Keďže sú tieto objekty vytvárané interne v prehliadači, v Brave sú jednoducho modifikované pred procesom vytvorenia v JavaScripte. Pri prístupe v rozšírení sú ale parametre týchto objektov read-only, nie je možné ich teda upravovať. Preto bol zvolený prístup vytvárania nových objektov podľa vzoru originálnych, ktorými sa tie pôvodné nahradia. Pôvodné metódy objektov *item*, *namedItem* a prípadne *refresh* sú vždy implementované, čiže sa objekty správajú ako tie originálne.
Táto ochrana má tri úrovne nastavenia a vlastnosti sú v úrovniach modifikované nasledovne:
Úroveň 0 - miernejšia ochrana:

- ***navigator.plugins***: v zozname plugin[15] objektov sú objekty nahradené novými, odkazy na *mimeType* sú aktualizované na nové objekty zo zoznamu *navigator.mimeTypes*, vlastnosť *name* je prebratá z pôvodného objektu a pri PDF pluginoch sú slová názvu pozmenené podľa slovníku a vlastnosti *name* a *description* sú rovnaké ako v pôvodným objektoch. Ďalej sú do zoznamu pridané dva náhodné pluginy, ich odkazy na *mimeType* odkazujú na nové *mimeType* objekty s náhodne vygenerovaným *description* a ostatnými prázdnymi vlastnosťami. Plugin vlastnosti *name*, *filename* a *description* sú pseudonáhodne vygenerované reťazce zvolenej dĺžky
- ***navigator.mimeTypes***: v zozname mimeType[16] objektov sú objekty nahradené novými, pri ktorých je odkaz *enabledPlugin* aktualizovaný na nový *plugin* objekt, ostatné vlastnosti sú rovnaké

Úroveň 1 - striktnejšia ochrana:

- **navigator.plugins**: pôvodný zoznam sa zahodí a vytvoria sa dva nové plugin objekty, ktoré sa pridajú do nového zoznamu. Pri týchto plugin objektoch odkazy na *mimeType* odkazujú na nové mimeType objekty s náhodne vygenerovaným *description* a ostatnými prázdnymi vlastnosťami. Plugin vlastnosti *name*, *filename* a *description* sú pseunáhodne vygenerované reťazce zvolenej dĺžky.

[12] https://developer.mozilla.org/en-US/docs/Web/API/WebGLActiveInfo
[13] https://developer.mozilla.org/en-US/docs/Web/API/WebGLShaderPrecisionFormat
[14] https://github.com/brave/brave-browser/issues/10597
[15] https://developer.mozilla.org/en-US/docs/Web/API/Plugin
[16] https://developer.mozilla.org/en-US/docs/Web/API/mimeType

- navigator.mimeTypes: pôvodný *mimeType* zoznam je nahradený prázdnym

Úroveň 2 - konzistentný odtlačok:

- **navigator.plugins**: pôvodný *plugin* zoznam je nahradený prázdnym
- navigator.mimeTypes: pôvodný *mimeType* zoznam je nahradený prázdnym

Táto ochrana je potom použitá v JavaScript Restrictor úrovniach 2 a 3, v oboch je využívaná úroveň 1, kde sú v zozname pluginov dva náhodné pluginy, prázdny mimeType zoznam. Ak bude v budúcnosti v prehliadači Chrome nakoniec upravené toto API, aby boli vracané prázdne zoznamy, tak sa odporúča nastaviť úroveň 2, alebo vypnúť túto časť ochrany.

5.5 enumerateDevices

Ako bolo spomenuté v 3.3.3, ochrana proti fingerprintingu v prehliadači Brave bola počas riešenia tejto práce nedokončená. Toto je práve jedno z API, ktoré zostalo kvôli neustálenému štandardu nedokončené. Pri implementácii sa táto práca držala momentálnej stabilnej verzie Brave a táto funkcionalita bola ďalej rozšírená. API sú obaľované nasledovne:
Úroveň 0 - miernejšia ochrana: Na tejto úrovni je ochrana implementovaná rovnako ako v Brave, a teda - na základe doménového hashu sú zariadenia v poli poprehadzované a vo výsledku sú v pseudonáhodnom poradí oproti pôvodnému. V prípade malého počtu pripojených zariadení (jeden mikrofón, jeden reproduktor) má takáto ochrana len malú účinnosť. Takúto ochranu možno taktiež obísť, ak fingerprinting skript po získaní zoznamu zariadenia usporiada podľa ľubovoľnej funkcie.
Úroveň 1 - striktnejšia ochrana: Na tejto úrovni sú do zoznamu zariadení pridané 0-4 falošné zariadenia typu *MediaDeviceInfo*. Tieto zariadenia majú náhodne určený typ - *videoinput*, *audioinput*, alebo *audiooutput*, ďalej sú vygenerované náhodné identifikátory skupiny *groupId* - ich dĺžky sa líšia podľa prehliadaču Firefox identifikátory majú 43 znakov a Chrome (prípadne Chromium prehliadače) 64 znakov. Odlišuje sa tiež prístup pri vlastnosti *deviceId*, pri Chrome je táto vlastnosť prázdny reťazec a pri Firefox sa generuje identifikátor, podobne ako pri *groupId*, tak sa zachováva konzistentnosť so správaním rôznych prehliadačov. Nakoniec je v zozname zariadení poprehadzované poradie zariadení, rovnako ako v úrovni 0 a zoznam je vrátený.
Úroveň 2 - konzistentný odtlačok: V tejto úrovni je zachovaná pôvodná funkcionalita, dostupná v predošlej verzii JavaScript Restrictor, čiže vrátenie prázdneho *promise* objektu. Aj keď by sa stránky nemali spoliehať výhradne na toto API, tak bolo testovaním zistené, že niektoré webové služby poskytujúce hovory, či videohovory pri vrátení prázdneho zoznamu zariadení nefunguje správne.

5.6 hardwareConcurrency

Toto API je obaľované v troch úrovniach, z čoho prvé dve úrovne sú založené na ochrane v Brave a tretia úroveň je zachováva pôvodnú funkcionalitu z JavaScript Restrictoru. Ochrana funguje tak, že nahradzuje pôvodnú hodnotu vracanú pri získavaní tejto vlastnosti. *Navigator.hardwareConcurrency* je read-only, čiže ju nejde jednoducho prepísať, preto je getter nahradený a v sebe volá funkciu, ktorá vracia hodnotu podľa úrovne ochrany. Generovanie a nahradenie sa vykonáva pri načítavaní stránky, vďaka čomu je hodnota konštantná na konkrétnej doméne, a rôzna naprieč doménami.

Obalená vlastnosť vracia nasledovné hodnoty:
Úroveň 0 - miernejšia ochrana: Vracia sa náhodná celočíselná hodnota medzi 2 a reálnou hodnotu získanou z vlastnosti pred obalením. Túto úroveň sa odporúča používať, ak má zariadenie menej, než 8 logických procesorov, aby nedošlo k znefunkčneniu stránok, ktoré na tom závisia. V prípade, že je reálna hodnota vyššia než 8, tak sa táto úroveň správa rovnako ako úroveň 1.
Úroveň 1 - striktnejšia ochrana: Vracia sa náhodná celočíselná hodnota medzi 2 a 8.
Úroveň 2 - konzistentný odtlačok: Vracia vždy 2. Toto je pôvodná funkcionalita z predošlej verzie JavaScript Restrictoru, kedy bola snaha prezentovať konštantný odtlačok, nemala by byť teda používaná v kombinácii s novou ochranou.

5.7 deviceMemory

Táto ochrana funguje podobne ako predošlá, snaží sa poskytovať pseudonáhodnú, ale valídnu hodnotu operačnej pamäte zariadenia, teda z množiny {0,25; 0,5; 1; 2; 4; 8 }. Vlastnosť *deviceMemory* je tiež nahradená funkciou, ktorá vracia rôzne hodnoty v závislosti na zvolenej úrovni:
Úroveň 0 - miernejšia ochrana: Vracia pseudonáhodnú hodnotu zo spomínanej množiny, medzi 0,25 a reálnou hodnotu získanou z vlastnosti pred obalením. Táto úroveň sa odporúča používať v prípade, že má zariadenie menej ako 8 GB operačnej pamäte. V prípade, že je reálna hodnota vyššia než 8, tak sa táto úroveň správa rovnako ako úroveň 1.
Úroveň 1 - striktnejšia ochrana: Vracia pseudonáhodnú hodnotu zo spomínanej množiny, medzi 0,25 a 8.
Úroveň 2 - konzistentný odtlačok: Vracia vždy 4. Toto je pôvodná funkcionalita z predošlej verzie JavaScript Restrictoru a nemala by byť používaná v kombinácii s novou ochranou.

5.7.1 User agent

Prehliadač Brave implementuje ochranu tejto časti tak, že pridáva náhodný počet medzier na koniec user agent reťazca a používa Chrome formát s najnovšou verziou Chrome prehliadaču.

Napriek tomu po dôkladnom prehodnotení nebola táto ochrana implementovaná do rozšírenia JavaScript Restrictor. Hlavným dôvodom bol momentálny stav user agent stringu, ako bolo aj spomínané na konci 2.2.6, kedy sa plánuje prechod na iný štandard, kvôli možnosti jednoduchého sledovania prehliadačov. Taktiež je JavaScript Restrictor používaný na viacerých platformách a bolo by teda možné iba pridať na koniec spomínané prázdne znaky, čo by mohlo zmiasť niektoré skripty vykonávajúce fingerprinting, no takúto ochranu veľmi ľahko obísť. Nejaká forma ochrany user-agent by mohla byť budúcim rozšírením tejto práce.

5.8 Identifikovateľnosť ochrany

V súvislosti s implementáciou je treba uvažovať aj nad jej slabými stránkami, poprípade spôsobmi, ako ochranu obísť. Jedným zo spôsobov, ako sa z pohľadu útočníka vyhnúť ochrane pred snímaním odtlačku je rozpoznať, o akú ochranu sa jedná (aký prehliadač, či rozšírenie) a ignorovať zdroje informácií, ktoré ochrana pokrýva. V prípade ochrany Brave a JSR je ich možné rozpoznať a dokonca rozlíšiť medzi sebou.

5.8.1 Rozpoznanie ochrany Brave a JSR

Pokúsiť sa rozpoznať takúto ochranu je možné rôznymi spôsobmi, možno analyzovať, či sa zmenil obsah obrázku pri volaní metód na získavanie dát z canvasu, čo je typicky nutné urobiť viac krát, aby sa prejavili rozdiely[17]. Ďalším API, ktoré možno jednoducho použiť je navigator.plugins, kedy majú falošné pluginy generované náhodné názvy, čo je možné rozpoznať. Najlepším rozhraním na identifikáciu je však WebGL, keďže niektoré návratové hodnoty sú modifikované o 1, čo produkuje atypické hodnoty a pri porovnaní so zoznamom očakávaných hodnôt by bolo možné jednoducho zistiť, že dochádza k úprave hodnôt.

Reálnu mieru identifikovateľnosti by bolo možné vyhodnotiť podrobným testovaním, pravdepodobne by bolo vhodné spomínané prístupy skombinovať.

Bolo by dokonca možné medzi sebou rozoznať ochranu Brave a JSR, rozdiely by sa prejavili napríklad v User agent, kedy Brave pridáva na koniec prázdne znaky, a tiež metódy CanvasRenderingContext2D.isPointInPath a CanvasRenderingContext2D.isPointInStroke sa správajú odlišne.

[17] https://pxlsfiddle.com/farbling.html

Kapitola 6

Testovanie

Táto kapitola sa venuje jednak vytváraniu testov a testovacej stránke, ale aj testovanie a porovnanie výslednej implementácie s prehliadačom Brave a konečné vyhodnotenie práce.

6.1 Testovacia stránka

Počas implementovania tejto práce bola upravovaná a rozširovaná pôvodná testovacia stránka, ktorá je súčasťou rozšírenia JSR. S pridaním ochranných prvkov boli teda na testovaciu stránku pridané časti demonštrujúce ochranné opatrenia, ktoré zároveň pomáhajú pri integračných testoch.

Pôvodná testovacia stránka obsahovala demonštrovanie ochrany canvasu v podobe plátna a funkcií na vykresľovanie elementov na náhodnú pozíciu, ďalej tu boli tlačidlá na získanie dát funkciou *canvas.toDataURL*. Táto funkcionalita bola ponechaná a je možné, že v ďalších verziách bude úplne odstránená, keďže ochrana canvasu už funguje iným spôsobom a táto demonštrácia je nedostatočná. Obaľovania canvas volaní sú demonštrované na troch plátnach (canvas elementoch) prvé vykresľuje obrázok zvolený v HTML, druhé plátno dáta z funkcie *getImageData* a tretie dáta z funkcie *toDataURL*. Používateľ tak môže ihneď vidieť zmeny, ktoré boli s dátami pri načítavaní vykonané, napríklad pri nahradení bielym obrázkom budú druhé a tretie plátno prázdne.

Pôvodný zámer bol taký, že URL dáta bude možno porovnať priamo v prehliadači a používateľovi povedať, či sa plátna líšia. Nastal však problém, pretože v prehliadači Firefox funkcia *toDataURL* kóduje dáta inak a to aj po nastavení maximálnej kvality. Z tohto dôvodu nemožno v prehliadači konzistentne porovnávať, či je výsledné plátno iné, pretože vo Firefoxe je vždy iné než originálne plátno. Riešením by mohlo byť využitie knižnice, alebo vytvorenie implementácie, ktorá by porovnala dve plátna a zvýraznila rozdiely.

Volania metód *CanvasRenderingContext2D.isPointInPath* a *CanvasRenderingContext2D.isPointInStroke* sú demonštrované na plátnach, ktoré menia farbu, ak tieto volania s polohou myši vrátia *true*. Pri zapnutej ochrane je teda pri pohybe myšou cez plátno vidno občasné prefarbenie.

WebGL canvas metódy sú na testovacej stránke demonštrované dvomi ďalšími canvasmi, jeden, ktorý zobrazuje výsledok metódy *toDataURL* a druhý výsledok *readPixels*. Výsledok *readPixels* je prevrátený, je nutné teda vykonať transformáciu dát, ktoré sa potom vložia do nového canvasu. Vytvorenie a inicializácia WebGL kontextu je urobené podobne ako na stránkach Browserleaks a Am I unique.

Ďalej sú v textovej podobe vypísané výsledky metódy *getParameter* pre VENDOR, UNMASKED_VENDOR, RENDERER, UNMASKED_RENDERER, VERSION, SHADING_LANGUAGE_VERSION. Ďalej sú ako zoznam zobrazené návratové hodnoty metódy *getShaderPrecisionFormat* pre všetky argumenty a tiež výsledok metódy *getSupportedExtensions*. Z praktického hľadiska do testovacej stránky neboli pridané všetky obaľované WebGL metódy a parametre, keďže by to bol jednak pridlhý zoznam a pre používateľa by nebol nijak zaujímavý – preto sú testované bez zobrazenia, priamo cez skripty v integračných testoch.

Výsledky metód pre AudioBuffer a Analyser node sú zobrazované v textovej podobe, výsledky *getChannelData* a *copyFromChannel* sú reprezentované súčtom všetkých hodnôt, ostatné metódy

getFloatFrequencyData, *getByteFrequencyData*, *getFloatTimeDomainData*, *getByteTimeDomainData* sú zobrazené ako prvých 40 položiek vráteného poľa.

6.2 Testy

Súčasťou vývoja bolo aj vytvorenie testov. Neboli implementované jednotkové testy, keďže ich využitie by bolo minimálne – všetky časti ochrany totiž fungujú iba pri spustení v prehliadači a jednotkové testy testujú funkcie samostatne. Testovanie tak bolo zamerané na ochranu ako celok a jej správne fungovanie, čiže boli vytvorené nové integračné testy a pôvodné testy boli rozšírené o nové testovacie prípady.

6.2.1 Integračné testy

Pri vytváraní testov táto práca naväzuje na prácu [6], kde pán Bednář podrobne rozoberá problematiku testovania a aj vytváranie automatizovaných testov pre JSR. Používa sa na to Selenium a drivery chromedriver a geckodriver. Tieto testy sú zamerané na jednotlivé obaľované API a kvôli spôsobu, akým sa výsledky podvrhujú bol zvolený prístup, kedy sú uložené namerané hodnoty bez zapnutého rozšírenia JavaScript Restrictor a tie sú potom porovnávané s hodnotami nameranými na konkrétnych úrovniach ochrany, pričom pri znáhodňovaní sa musia líšiť, alebo sú očakávané hodnoty z rozsahu hodnôt.

Počas vývoja vznikla aj nová úroveň JSR, ktorá však nie je momentálnej verzii zahrnutá do rozšírenia. Táto testovacia úroveň sa vytvorí manuálne s použitím Selenia pred začiatkom integračného testovania. Táto úroveň nastavuje ochranu canvasu znáhodnením, rovnako aj ochranu webaudia a WebGL, podvrhovanie falošných pluginov, media zariadení a znáhodnenie hardvérových vlastností, ostatné opatrenia sú aktívne rovnako ako v úrovni 2. V budúcnosti je podobný prístup možný použiť aj na testovanie rôznych kombinácií nastavení, ktoré však nemajú vlastnú úroveň.

Canvas

Pri canvase integračné testy zahŕňajú všetky obaľované metódy na získavanie dát z canvasu, v pre každú metódu je napísaný jeden test, ktorý porovnáva vrátené dáta s hodnotami nameranými bez rozšírenia, ktoré sa v prípade zapnutej ochrany musia líšiť. Taktiež sa odchytávajú výnimky v prípade zlyhania operácií. Metóda *OffscreenCanvas.convertToBlob* nie jezahrnutá, ale mala by sa správať rovnako ako pri normálnom canvase. Canvas metódy *isPointInStroke* a *isPointInPath* nie sú v momentálnej verzii testov zahrnuté. No v budúcej verzii by mohli testy fungovať tak, že by boli metódy volané opakovane s rovnakými súradnicami – až niekoľko sto krát, aby sa znížila pravdepodobnosť náhodného zlyhania – a podľa výsledkov by sa vyhodnotilo, či boli výsledky správne podvrhnuté, je však pravda, že by existovala malá šanca, že test zlyhá aj keď bude ochrana fungovať správne.

WebGL

WebGL testy obsahujú testy odmaskovaného rendereru a vendora, ktoré v prípade zapnutej WebGL ochrany očakávajú iné výsledky oproti výsledkom z prehliadaču bez JSR, keďže sú v ochrane tieto vlastnosti nahradené vygenerovanými reťazcami. Bez zapnutej ochrany očakávajú testy rovnaké výsledky. Testuje sa metóda *getParameter* s všetkými vstupnými argumentami, ktoré v implementácii obaľujeme. Pri zapnutí úrovne 0 sa očakáva, že výsledky pri argumentoch, kedy vracia metóda číselné hodnoty sú rovnaké, alebo sa líšia o 1. Pri vypnutej ochrane sa sa očakávajú rovnaké hodnoty ako bez JSR. Pri zapnutej úrovni 1 sa pri obalených argumentoch očakáva iná hodnota ako bez JSR. Inak sa očakáva rovnaká hodnota.

Ďalej sa testujú metódy pre WebGL canvas - *toDataURL* a *readPixels*. Obe tieto metódy sa volajú nad originálnym canvasom na testovacej stránke a musia mať pri zapnutej ochrane odlišné výsledky ako výsledky bez JSR a rovnaké pri vypnutej. Tiež je tu odchytávanie výnimky v prípade zlyhania operácie.

Samostatne sa testuje sa aj *getShaderPrecisionFormat*, kde sa prejde zoznam všetkých vstupných argumentov a porovnáva sa výsledok. Pri zapnutej ochrane musia byť objekty v zozname prázdne, pričom pri vypnutej ochrane musí byť zoznam rovnaký ako bez JavaScript Restrictoru.

Niektoré obalené WebGL metódy neboli zahrnuté do integračných testov a to najmä preto, že pri zapnutí úrovne 1, kedy mnoho týchto metód vracia spodné hodnoty, a teda *null*, prázdne zoznamy a podobne, dochádza k čiastočnému znefunkčneniu WebGL a tým pádom nie je možné korektne porovnávať výsledky (očakávať chyby by bolo nekorektné, keďže by sa nepotvrdilo správne fungovanie).

Audio

Web audio ochrana sa testuje v sade testov pre každú metódu spomínanú v implementácii, časti 5.2. Tieto metódy sú testované na výsledkoch, ktoré sú zobrazované v testovacej stránke. Odtiaľ sa čítajú hodnoty v podobe reťazcov a porovnávajú sa s očakávanými hodnotami.

Navigator

Boli upravené očakávané hodnoty pre *navigator.deviceMemory* a *navigator.hardwareConcurrency*, kedy je momentálne očakávaný rozsah valídnych hodnôt, namiesto jednej hodnoty ako to bolo pôvodne, no stále je zachovaný prípad pre úroveň, kedy je rozsah jedna konkrétna hodnota. Ďalej boli pridané testy pre *navigator.plugins* a *navigator.mimeTypes*, ktoré musia pri zapnutí ochrany vracať iné výsledky ako bez JSR.

Výsledky

Po napísaní integračných testov bolo rozšírenie opakovane testované a boli odhalené chyby, ktoré boli počas implementácie prehliadnuté, a ktoré spôsobovali zlyhávanie testov - napríklad chýbajúca implementácia metódy pri falošnom objekte, pridanie WebGL2 kompatibility a podobne.

Pri verzii, ktorá bola v rámci práce zakomponovaná do rozšírenia zlyhali niektoré testy, a to pri úrovni 3 na prehliadači Chrome, kedy ide pravdepodobne o chybu driveru, o ktorej sa vedelo aj v minulosti a je spomenutá v dokumentácii projektu.

6.3 Porovnanie výslednej ochrany

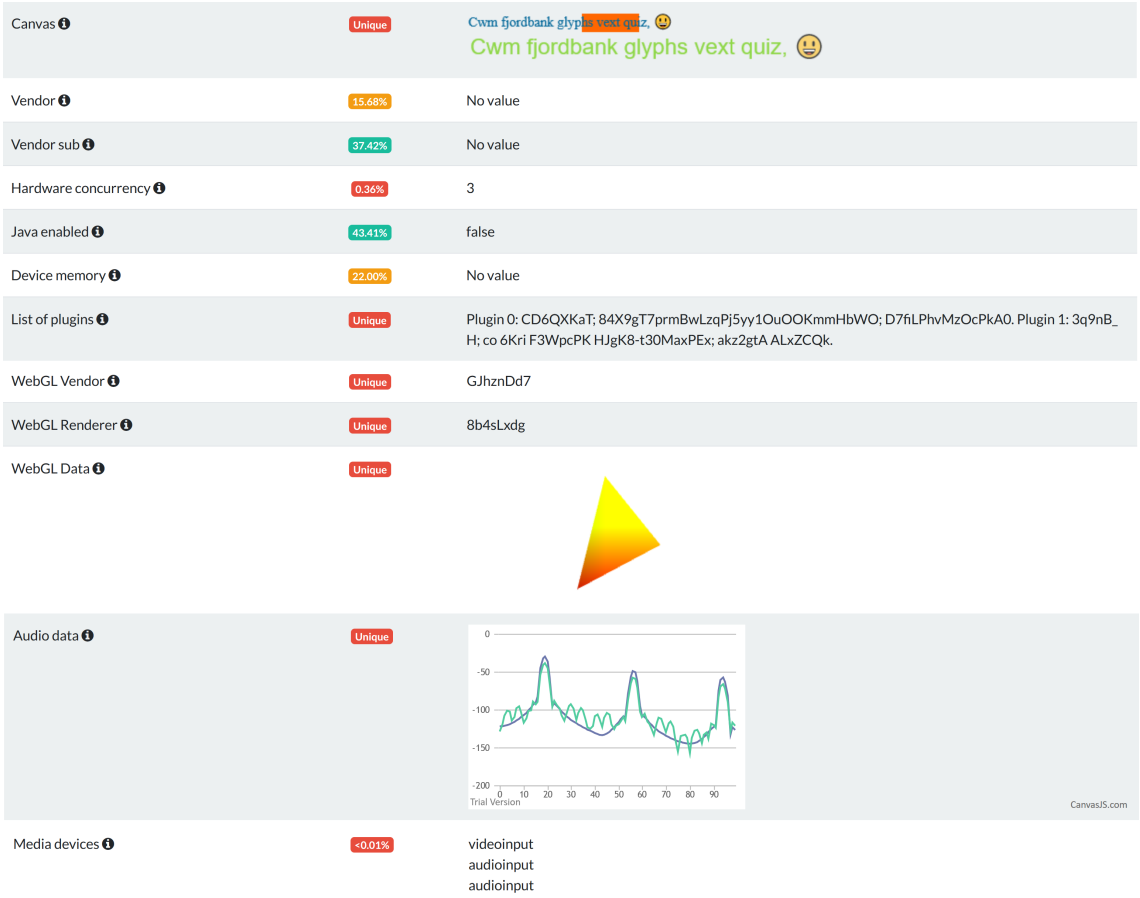
Táto časť obsahuje porovnanie výslednej implementácie ochrany proti snímaniu odtlačku prehliadaču v rozšírení JavaScript Restrictor použité v prehliadačoch Chrome 90.0.4430.93 (oficiálna zostava) (64-bitová verzia) a Mozilla Firefox 78.10.1esr (64-bit) s ochranou poskytovanou prehliadačom Brave. Porovnanie bolo demonštrované na webových službách poskytujúcich snímanie odtlačku.

Testy na nasledujúcich stránkach boli opakované aspoň desať krát, aby sa potvrdilo, že sú výsledky snímania konzistentné. Následne boli vybrané jeden, alebo dva odtlačky ako príklad - ale tvar bol pri všetkých testoch veľmi podobný. Úroveň použitá na tieto testy bola rovnaká ako tá, ktorá sa manuálne vytvorí pri integračnom testovaní, keďže je najpodobnejšia ochrane v Brave.

6.3.1 Am I unique

Stránka `https://amiunique.org/` poskytuje testovanie najväčšieho množstva prvkov používaných na získavanie odtlačku prehliadaču, bola vytvorená pri článku [28]. Okrem rozhraní zahrnutých v našej ochrane táto stránka pokrýva aj mnohé ďalšie, ako napríklad fonty, časovú zónu, rozlíšenie okna, analýzu pripojenia a monoho iného. Napriek veľkému rozsahu sa počas testovania podarilo demonštrovať, že časti pokrývané ochranou v JavaScript Restrictore úspešne prezentuje každú reláciu (alebo obnovenie rozšírenia) unikátne výstupy. Odtlačky sa však v tejto službe ďalej neanalyzujú a je možné, že pri odignorovaní častí, pri ktorých JSR klame by bol počet identifikujúcich bitov stále dostatočný.

Nižšie, na obrázku 6.1 je vidno, že zoznam pluginov aj všetky WebGL položky sú unikátne v celej sade odtlačkov, počet logických procesorov je tiež podvrhnutý ale kvôli malej množine valídnych hodnôt nie je unikátna, operačná pamäť ako štandardne vo Firefoxe nemá hodnotu a media zariadenia majú iné poradie.



Obr. 6.1: Odtlačok pre Firefox na Am I unique, vybrané časti

V Brave ani v Chrome s JSR nie je dostupný údaj Audio data, pretože inicializáciu nemožno vykonať cez skript, vo Firefoxe však takáto inicializácia funguje a odtlačok teda tento výsledok obsahuje.

Ako je vidieť na obrázku 6.1 v časti Audio data, zvuková stopa sa javí ako unikátna, čo bol výsledok pri každom teste. To napriek tomu, že pri pohľade na vizualizáciu na obrázku je signál na nerozoznanie od toho bez zapnutého rozšírenia.

Pri porovnaní s Brave a Chrome sú teda patrné isté odlišnosti, ktoré by mohli byť použité k lepšej identifikovateľnsti typu prehliadaču. Pluginy vo Firefoxe vracajú normálne prázdne pole, a Chrome nie, tým pádom po aplikácii ochrany bude v Chrome viac pluginov, a vo Firefoxe len dva. Stojí za zváženie teda pri pluginoch používať úroveň ochrany 1, ktorá vracia vždy dva náhodné pluginy. Ďalej sú rozdiely patrné pri deviceMemory, kedy Firefox túto vlastnosť neobsahuje. A napokon už spomínané Audio data, ktoré sú kvôli inicializácii v Chrome nedostupné.

6.3.2 Audio

Keďže snímanie audio odtlačku na Am I unique je inicializované skriptom, čo spôsobuje nefunkčnosť na Chrome a Brave, boli vykonané testy aj na stránke `https://audiofingerprint.openwpm.com`,

ktorá je určená špeciálne na testovanie audio a font fingerprinting. Fonty budeme v testoch teda ignorovať, keďže protiopatrenia proti ich snímaniu nie sú prítomné ani v Brave, ani v JSR.

Výsledky ukázali rovnaké správanie čo sa týka výsledných odtlačkov pri všetkých testovaných prehliadačoch, príklady výsledkov sú dostupné v prílohe A.3, pri podrobnejšom skúmaní si možno všimnúť drobné odlišnosti medzi prehliadačmi.

Zaujímavé bolo namerané spomalenie (spomalenie 1,4 až 11 krát pri opakovaných testoch, avšak len v prvej časti snímania odtlačku, ostatné časti trvali podobne dlho) - prehliadače so zapnutým JavaScript Restrictorom vykonávali test niekoľkonásobne dlhšie ako Brave, avšak pri veľmi podobnom teste na audio odtlačok na testovacej stránke JSR nebolo takéto spomalenie zaznemenané. Pri analýze spúšťaných skriptov bolo zistené, že sú funkcie vykonávané stovky tisíc krát, je teda zrejmé, že aj malé spomalenie spôsobené manipuláciou dát v JavaScripte zapríčiňuje pri obrovských objemoch dát značné spomalenie, ktoré treba brať do úvahy.

6.3.3 Cover Your Tracks

Cover Your Tracks[1] je webová stránka poskytujúca testovanie odtlačku prehliadaču s možnosťou použitia reálnej reklamnej/tracking spoločnosti, testuje tiež odtlačky na rôznych doménach, keďže používa presmerovania. Historická verzia bola použitá v článku [18] z roku 2010, ktorý je jedným z najvplyvnejších publikácií v tejto oblasti. V minulosti sa táto služba volala Panopticlick, čo je známejší názov, no so zmenou mena prišiel aj vylepšený systém snímania odtlačkov.

Brave level Balanced:

BROWSER PLUGIN DETAILS: randomized by first party domain
HASH OF CANVAS FINGERPRINT: randomized by first party domain
HASH OF WEBGL FINGERPRINT: randomized by first party domain
WEBGL VENDOR & RENDERER: Google Inc. (Intel) ANGLE (Intel, Intel(R) HD Graphics 5500Direct3D11 vs_5_0 ps_5_0, D3D11-20.19.15.5107)
AUDIOCONTEXT FINGERPRINT: randomized by first party domain
HARDWARE CONCURRENCY: 3
DEVICE MEMORY (GB): 1

Firefox s JSR level 4:

Browser Plugin Details: randomized by first party domain
Hash of canvas fingerprint: randomized by first party domain
Hash of WebGL fingerprint: randomized
WebGL Vendor & Renderer: h6qO8Zok~D0Y-aq4E
AudioContext fingerprint: randomized by first party domain
Hardware Concurrency: randomized
Device Memory (GB): N/A

Chrome s JSR level 4:

BROWSER PLUGIN DETAILS: randomized by first party domain
HASH OF CANVAS FINGERPRINT: randomized by first party domain
HASH OF WEBGL FINGERPRINT: randomized
WEBGL VENDOR & RENDERER: vtc0rTFm~uUB6sTY0
AUDIOCONTEXT FINGERPRINT: randomized by first party domain
HARDWARE CONCURRENCY: 2
DEVICE MEMORY (GB): 2

[1] https://coveryourtracks.eff.org/

Na týchto príkladoch možno vidieť, že služba *Cover your tracks* dokáže s pomerne veľkou presnosťou rozoznať, či je odtlačok, alebo časť odtlačku náhodne podvrhovaná. Pri celkovom hodnotení bol výsledok vždy buď – unikátny odtlačok, alebo odtlačok znáhodnený first-party doménou – čo naznačuje správne fungovanie ochrany implementovanej v tejto práci, ktorá sa snaží prezentovať vždy unikátny odtlačok. Jediný znateľný rozdiel pri pokrytých API oproti Barve možno vidieť pri časti WebGL Vendor & Renderer, kedy Brave tento údaj znáhodňuje až na najvyššej úrovni. Okrem toho sú v tejto službe výsledky snímania odtlačku takmer ekvivalentné pre Brave ochranu a pre ochranu implementovanú v tejto práci.

6.3.4 Fingerprint.js

Fingerpring.js[2] je komerčne dostupná služba pre stránky, ktoré chcú predísť podozrivým prístupom do účtov ich klientov. Táto služba identifikuje používateľské prehliadače na základe rôznych faktorov, ako je IP adresa, cookies, cache a podobne. Cez ich platenú službu nerobia cross-domain sledovanie, iba monitorovanie prístupov na jednej doméne, no ich open source knižnicu možno použiť, a je aj často používaná, na vytvorenie systémov na sledovanie prehliadačov naprieč doménami. Narozdiel od predošlých stránok, ktoré mali skôr informatívny charakter sa jedná o reálne nasadenú službu. Je pravda, že po zakúpení a používaní na stránkach sa nevykonáva cross-domain sledovanie, no ponúkajú aj open source knižnicu, s pomocou ktorej je možné implementovať vlastný fingerprinting systém.

Na webe tiež ponúkajú demo, pomocou ktorého možno vyskúšať túto službu. Ako však bolo spomínané, identifikácia nie je robená výhradne na základe odtlačku prehliadaču, odtlačok sa použije až pri zlyhaní jednoduchších prístupov. Preto je nutné vykonať kroky naviac, aby sme zaistili rozpoznanie na základe odtlačku.

Pri prvotnom testovaní bol pri každom testovaní rozpoznaný rovnaký identifikátor v danom prehliadači, teda Brave, Chrome aj Firefox mali každý svoj identifikátor, ktorý bol pri všetkých nastaveniach ochrany rovnaký. Preto boli na ďalšie testy vykonané tieto opatrenia:

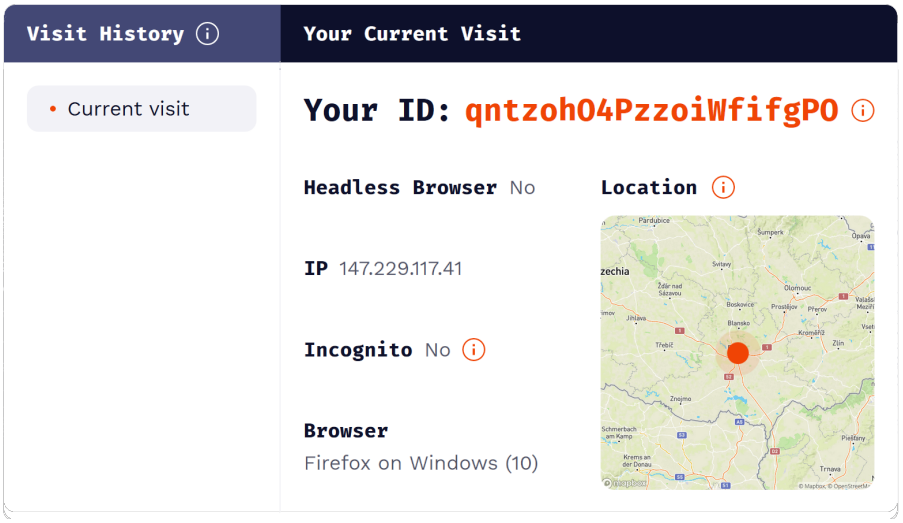
- Pri každom teste boli vymazané cookies a cache prehliadaču.
- Bola spustená nová relácia.
- Bola použitá VPN na skrytie IP a polohy.

Po testovaní rôznych kombinácií týchto nastavení boli pozorované nasledujúce vzorce správania:

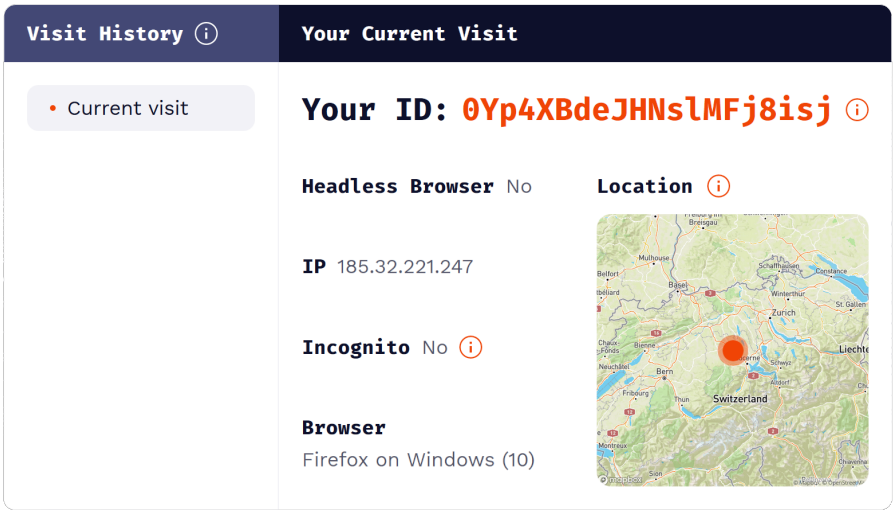
- Použitie VPN a prepínanie IP adries samo o sebe nezaručí nový identifikátor
- Vymazanie cookies a cache samo o sebe nezaručí nový identifikátor
- Reštartovanie prehliadaču samo o sebe nezaručí nový identifikátor
- Obnovenie rozšírenia, obnovenie VPN a vymazanie cache a cookies zaručuje nový identifikátor
- Vymazanie cache a cookies a reštartovanie prehliadaču zaručuje nový identifikátor a to aj bez použitia VPN, preto je toto najefektívnejšia možnosť
- Vymazanie cache a cookies, reštartovanie prehliadaču a použitie VPN niekedy viedlo k novému identifikátoru aj bez zapnutého rozšírenia

Vďaka týmto testom vieme povedať, že novo implementovaná ochrana pomáha vytvárať jedinečné odtlačky a to aj pri použití rovnakej IP adresy, je nutné však vymazávať cache, čo by mohlo byť súčasťou ďalších verzií v záujme ochrany pred fingerprintjs službou. Pri zapnutej ochrane bol ale pri každom testovaní po vykonaní spomínaných krokov každému prehliadaču priradený iný identifikátor bez histórie návštev, tým pádom je zrejmé, že ochrana proti snímaniu odtlačku funguje správne. Príklad môžeme vidieť pri porovnaní obrázkov 6.2 a 6.3. Ďalšie príklady z tohto testovania možno nájsť v prílohe A.2.

[2] https://fingerprintjs.com/



Obr. 6.2: Odtlačok pre Firefox s rozšírením JSR úrovňou 4 na fingerprint.js demo



Obr. 6.3: Odtlačok pre Firefox s rozšírením JSR úrovňou 4 na fingerprint.js demo, po reštartovaní prehliadaču a zmazaní dát

6.3.5 Testovanie na bežných stránkach

Dôležité je aj otestovať, či nemá implementovaná ochrana nepriaznivý vplyv na bežné stránky, ktoré väčšina používateľov internetu navštevuje. Preto bol vybraný zoznam stránok: Facebook, Google, Gmail, Youtube, Twitter, Spotify. Ďalej bol vybraný zoznam stránok, ktoré síce nie sú také populárne, no využívajú API pre WebGL, canvas a podobne, čiže je možné otestovať funkčnosť týchto API na rôznych úrovniach: jitsi[3], diagrams[4], thelabberkeley[5].

[3] https://meet.jit.si/
[4] https://app.diagrams.net/
[5] https://thelabberkeley.com/

Názov stránky	Správanie na úrovniach	
	Balanced	Maximum
Facebook	Plne funkčná	Plne funkčná
Google	Plne funkčná	Plne funkčná
Gmail	Plne funkčná	Plne funkčná
Youtube	Plne funkčná	Plne funkčná
Twitter	Plne funkčná	Plne funkčná
Spotify	Plne funkčná	Plne funkčná
Jitsi	Plne funkčná	Plne funkčná
Meet	Plne funkčná	Plne funkčná
Diagrams	Plne funkčná	Plne funkčná
Thelabberkeley	Plne funkčná	Nefunkčná (nejde načítať, chyby WebGL)

Tabuľka 6.1: Brave Verzia 1.23.75 Chromium: 90.0.4430.93 (oficiálna zostava) (64-bitová verzia)

Názov stránky	Správanie na úrovniach		
	2	3	4
Facebook	Plne funkčná	Plne funkčná	Plne funkčná
Google	Plne funkčná	Plne funkčná	Plne funkčná
Gmail	Plne funkčná	Plne funkčná	Plne funkčná
Youtube	Plne funkčná	Nefunkčná (nejdú prehrávať videá)	Plne funkčná
Twitter	Plne funkčná	Nefunkčná (niektoré obrazovky nefungujú, chyba s ArrayBufferView)	Plne funkčná
Spotify	Plne funkčná	Nefunkčná (niektoré obrazovky nefungujú, chyba s ArrayBufferView)	Plne funkčná
Jitsi	Plne funkčná	Nefunkčná (niektoré obrazovky nefungujú, chyba s ArrayBufferView)	Plne funkčná
Meet	Plne funkčná	Nefunkčná	Plne funkčná
Diagrams	Plne funkčná	Nefunkčná (range error)	Plne funkčná
Thelabberkeley	Nefunkčná (nejde načítať, chyby WebGL)	Nefunkčná (nejde načítať, chyby WebGL)	Plne funkčná

Tabuľka 6.2: JavaScript Restrictor na Chrome 90.0.4430.93 (oficiálna zostava) (64-bitová verzia)

Názov stránky	Správanie na úrovniach		
	2	3	4
Facebook	Plne funkčná	Plne funkčná	Plne funkčná
Google	Plne funkčná	Plne funkčná	Plne funkčná
Gmail	Nefunkčná (Date)	Nefunkčná (Date)	Nefunkčná (Date)
Youtube	Plne funkčná	Nefunkčná (nejdú prehrávať videá)	Plne funkčná
Twitter	Nefunkčná (Date)	Nefunkčná (Date)	Nefunkčná (Date)
Spotify	Plne funkčná	Nefunkčná (niektoré obrazovky nefungujú, chyba s ArrayBufferView)	Plne funkčná
Jitsi	Plne funkčná	Nefunkčná (niektoré obrazovky nefungujú, chyba s ArrayBufferView)	Plne funkčná
Meet	Plne funkčná	Nefunkčná	Plne funkčná
Diagrams	Plne funkčná	Nefunkčná (Dom exception)	Plne funkčná
Thelabberkeley	Nefunkčná (nejde načítať, chyby WebGL)	Nefunkčná (nejde načítať, chyby WebGL)	Plne funkčná

Tabuľka 6.3: JavaScript Restrictor na Mozilla Firefox 78.10.1 esr (64-bit)

6.4 Vyhodnotenie

Po prehodnotení predošlých testov sa rozšírenie JavaScript Restrictor javí ako účinné riešenie na ochranu pred snímaním odtlačku prehliadaču, vo väčšine ohľadov porovnateľné s ochranou v prehliadači Brave. Niektoré časti ochrany, ktoré Brave obsahuje nie sú implementované, ale jedná sa skôr o diskutabilné prvky, ako je user agent a časti, ktorých vývoj stále prebieha. Celkovo však výsledky naznačujú dostatočnú ochranu a to aj pri komerčne používaných službách.

Testovanie na bežných stránkach ukázalo, že nová ochrana s miernejším nastavením nespôsobuje poškodenie stránok. V tabuľke 6.3 pri úrovni 4 možno vidieť, že pri prehliadači Firefox ESR na niektorých stránkach síce spôsobilo obaľovanie Date objektu nefunkčnosť, toto však nebolo súčasťou tejto práce a jedná sa zrejme o chybu, ktorá bude v budúcnosti opravená. Po vypnutí ochrany Date objektu fungovali všetky stránky. Striktná úroveň 3, často spôsobuje nefunkčnosť kvôli ArrayBuffer chybám, ktoré tiež nie sú súčasťou novej ochrany. Nefunkčnosti WebGL na úrovniach 2 a 3 sú očakávané, keďže sú použité striktnejšie protiopatrenia - podobne ako pri Brave na úrovni maximum.

Testovanie odhalilo aj nedostatky momentálnej implementácie JavaScript Restrictoru, konkrétne obaľovanie objektov ArrayBuffer a sharedArrayBuffer spôsobuje nefunkčnosť niektorých stránok (napríklad `https://audiofingerprint.openwpm.com`). Na rovnakej stránke bolo zaznamenané spomalenie spôsobené aplikovaním filtrov na dáta, ktoré nie je pri bežnom používaní kritické, no v špecifických situáciách môže znamenať úplnú nefunkčnosť prehliadaču.

Podľa [25] sú API a funkcie, ktoré pokrývame sú často používané vo fingerprinting skriptoch. Tento výskum samozrejme hovorí aj o mnohých ďalších rozhraniach, ktoré zatiaľ pokryté nie sú, a v rámci budúcich prác by bolo vhodné niektoré zahrnúť.

Pri analýze knižnice fingerprint.js[6] sa ukázalo, že aj keď väčšina rozhraní pokrývaných novo implementovanou ochranou proti snímaniu odtlačku v JavaScript Restrictore sú v tejto knižnici používané, knižnica používa aj veľmi veľa ďalších. Nachádzajú sa tam napríklad skripty, ktoré využívajú na identifikáciu CSS vlastnosti. Takže aj keď bola momentálne ochrana dostatočná, aby zmiatla fingerprint.js, je možné, že budúcimi vylepšeniami tejto knižnice sa situácia obráti.

6.5 Nasadenie

Vývoj prebiehal na GitHub repozitári[7], ktorý bol vytvorený pomocou forku hlavného repozitáru projektu[8]. Celý proces vývoja, všetky zmeny a commity sú zaznamenané *gitom* a v prípade potreby je možné opraviť problémy, alebo vrátiť zmeny. Pridaná funkcionalita tejto práce bola do hlavného projektu zahrnutá v dvoch fázach, pričom každá mala vlastný pull request. Prvý pull-request[9] obsahoval základné časti implementácie, ako je generovanie a systém ukladania hashov, časti canvas, audio a WebGL ochrany a dokumentáciu k týmto častiam.

Druhý pull-request[10] obsahoval testovaciu stránku a integračné testy, dokončenie WebGL ochrany, ostatné časti ochrany ako pluginy, media devices a podobne. Obsahoval tiež opravy malých chýb, ktorý boli odhalené počas systematického testovania.

Prvý pull-request bol zakomponovaný do hlavného vydania a preto je v čase odovzdávania práce súčasťou rozšírenia dostupného na obchodoch Mozilla addons[11] a obchod Chrome[12]. Druhý pull-request by mal byť do rozšírenia zakomponovaný čoskoro po odovzdaní práce. Neskôr by mala prísť aj verzia upravujúca štruktúru úrovní ochrany v JSR.

Vzhľadom na výsledky testovania bude štruktúra úrovní v JavaScript Restrictore prerobená. Testovacia úroveň 4 by mala nahradiť momentálnu úroveň 2 a úroveň 3 bude možno mierne poupravená.

6.6 Budúci vývoj

Jednou z oblastí, ktorú by bolo možné v budúcnosti pomerne jednoducho vylepšiť je canvas ochrana v úrovni 1. Prezentovanie konštantne bieleho obrázku je pri momentálnej implementácii ochrany kontraproduktívne a ak by bola snaha prezentovať obrázok, ktorý nijak nesúvisí s pôvodným canvasom, tak by bolo lepšie použiť biely šum, podobne ako pri audio ochrane úrovni 1 popísanej v časti 5.2 . Takáto ochrana by fungovala veľmi podobne, obsah pôvodného canvasu by sa odignoroval a pomocou posuvného registra s lineárnou spätnou väzbou, alebo prípadne generátora náhodných čísiel by sa vygeneroval biely šum konštantný pre konkrétnu doménu. Problém by mohol nastať, ak by fingerprinting skripty implementovali rozpoznanie veľmi nízkych hodnôt a ignorovali odtlačok, tak ako to robí demonštrácia na Browserleaks WebGL obrázku[13], čo však funguje aj pri momentálnej implementácii podvrhovania bieleho obrázku.

Ďalšou oblasťou budúceho zlepšovania je úprava histórie navštívených domén. Momentálne sú vytvárané a ukladané hashe pre každú doménu, čo síce funguje, no nie je to úplne efektívne čo sa týka pamäte, ani výkonu, a môže dochádzať k nechcene rôznym odtlačkom na rôznych poddoménach jednej domény. Ideálne by bolo v budúcej verzii implementovať podobný prístup ako má Brave a teda, jeden doménový hash pre eTLD+1 [26]. Toto by mohlo byť implementované za pomoci zoznamu Public Suffix List ako bolo už spomínané v 5. Potom by bolo nutné porovnať pamäťové a časové

[6] https://github.com/fingerprintjs/fingerprintjs
[7] https://github.com/xsvanc06/jsrestrictor
[8] https://github.com/polcak/jsrestrictor
[9] https://github.com/polcak/jsrestrictor/pull/79
[10] https://github.com/polcak/jsrestrictor/pull/99
[11] https://addons.mozilla.org/cs/firefox/addon/javascript-restrictor/
[12] https://chrome.google.com/webstore/detail/javascript-restrictor/ammoloihpcbognfddfjcljgembpibcmb
[13] https://browserleaks.com/webgl

nároky oboch riešení a vybrať to optimálnejšie.

Zlepšenie testovacej stránky by mohlo byť tiež jednou z tém ďalšieho pokračovania tejto práce. Momentálna testovacia stránka plní účel ako nástroj na integračné testy, no ako demonštrácia funkčnosti či nefunkčnosti ochrany proti získavaniu odtlačku prehliadača funguje len čiastočne. Budúce zlepšenie by malo spočívať v lepšej reprezentácii a vizualizácii jednotlivých častí, napríklad zobrazenie rozdielov medzi canvasmi pred a po volaní metód, zobrazenie rozdielov v audio signále a podobne. Tiež by bolo vhodné každú časť reprezentovať ako odtlačok v podobe hashu, následne tieto hashe vhodne zobrazovať a ideálne porovnávať hashe naprieč reláciami. Testovaciu stránku by bolo tiež vhodné umiestniť na dve domény, podobne ako funguje testovanie fingerprinting ochrany v Brave[14].

Jednou z oblastí, kde nie je ustálený štandard sú API pre senzory. Momentálne je ich možné v niektorých prehliadačoch používať na získavanie odtlačkov [16]. Preto, ak nebude ochrana proti takémuto zneužívaniu týchto rozhraní v prehliadačoch dostatočná, bolo by vhodné ju implementovať do JSR, ak to bude možné.

[14]Brave farbling testovacie stránky: https://dev-pages.brave.software/farbling.html, https://dev-pages.bravesoftware.com/farbling.html

Kapitola 7

Záver

Cieľom tejto práce bolo analyzovať opatrenia proti získavaniu odtlačku implementovanú v prehliadači Brave, preto je v prvých kapitolách priblížená problematika získavania odtlačku prehliadača. Tiež sú prezentované prístupy a riešenia ochrany pred snímaním odtlačku prehliadaču, a špeciálne tie, ktorých myšlienky ochrana v Brave čerpá.

Po zasadení do kontextu bolo popísané rozšírenie JavaScript Restrictor – najmä prvky ochrany proti získavaniu odtlačku. Ďalej práca analyzovala ochranu v prehliadači Brave, jej účinnosť a spôsob fungovania, a bola potom porovnaná s tou v JavaScript Restrictore.

Následne bol popísaný navrhovaný postup prenosu opatrení a tiež správanie ochrany v navrhnutých úrovniach. Podľa navrhnutého postupu bola ochrana proti snímaniu odtlačku prehliadaču po častiach implementovaná. Bola vylepšená testovacia stránka rozšírenia a vytvorené integračné testy, ktoré testovaciu stránku používajú.

Po dokončení implementácie bola účinnosť implementovanej ochrany testovaná na dostupných službách poskytujúcich demonštračné snímanie odtlačku prehliadaču, a výsledky boli vyhodnotené. Testy na spomínaných službách ukázali, že ochrana funguje správne a je podobne účinná ako ochrana v Brave, ktorá bola predlohou tejto práce. Nová implementácia je účinnejšia, ako pôvodná ochrana v JSR, a to aj preto, že nový prístup funguje fundamentálne odlišne, ale aj preto, že je nová ochrana rozsiahlejšia. Ochrana bola testovaná tiež na bežných stránkach a stránkach využívajúcich špecifické API, pričom bolo potvrdené, že nová ochrana nespôsobuje nefunkčnosť stránok a je teda vhodná na bežné používanie.

Nová ochrana bola z časti integrovaná do momentálnych úrovní ochrany JSR, a je súčasťou vydanej verzie. Je však v pláne úrovne prerobiť a používať novo implementované prvky ochrany ako východzie.

Nakoniec bolo prezentovaných niekoľko smerov, ktorými by sa mohol uberať postup ďalšieho vývoja nadväzujúceho na túto prácu, ako aj oblasti, v ktorých by bolo možné momentálnu implementáciu vylepšiť.

Literatúra

[1] Adobe flash player : Security vulnerabilities. URL https://www.cvedetails.com/vulnerability-list/vendor_id-53/product_id-6761/Adobe-Flash-Player.html.

[2] Intent to ship: Return empty for navigator.plugins and navigator.mimetypes. URL https://groups.google.com/a/chromium.org/g/blink-dev/c/bbxAGu90LgM.

[3] Opinion 9/2014 on the application of directive 2002/58/ec to device fingerprinting, 9 2014. URL https://ec.europa.eu/justice/article-29/documentation/opinion-recommendation/files/2014/wp224_en.pdf.

[4] Adobe flash player eol general information page, 2020. URL https://www.adobe.com/products/flashplayer/end-of-life.html.

[5] David Bau. davidbau/seedrandom. URL https://github.com/davidbau/seedrandom.

[6] Martin Bednář. Automatické testování projektu javascript restrictor. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2020. URL https://www.fit.vut.cz/study/thesis/22376/.

[7] Leyla Bilge and Tudor Dumitraş. Before we knew it: An empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 833–844, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450316514. doi: 10.1145/2382196.2382284. URL https://doi.org/10.1145/2382196.2382284.

[8] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. User tracking on the web via cross-browser fingerprinting. In Peeter Laud, editor, *Information Security Technology for Applications*, pages 31–46, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-29615-4.

[9] Brave. Brave, fingerprinting, and privacy budgets, 11 2019. URL https://brave.com/brave-fingerprinting-and-privacy-budgets/.

[10] Brave. Brave launches next-generation browser that puts users in charge of their internet experience with unmatched privacy and rewards, 1 2020. URL https://brave.com/brave-launches-next-generation-browser/.

[11] Brave. Brave improves its ad-blocker performance by 69x with new engine implementation in rust, 1 2020. URL https://brave.com/improved-ad-blocker-performance/.

[12] Brave. What's brave done for my privacy lately? episode #2: Third-party cosmetic filtering, 2 2020. URL https://brave.com/privacy-updates-2/.

[13] Brave. What's brave done for my privacy lately? episode #3: Fingerprint randomization, 3 2020. URL https://brave.com/privacy-updates-3/.

[14] Brave. What's brave done for my privacy lately? episode #4: Fingerprinting defenses 2.0, 5 2020. URL https://brave.com/privacy-updates-4/.

[15] Yinzhi Cao, Song Li, and Erik Wijmans. (cross-)browser fingerprinting via os and hardware level features. 01 2017. doi: 10.14722/ndss.2017.23152.

[16] Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. The web's sixth sense: A study of scripts accessing smartphone sensors. pages 1515–1532, 10 2018. doi: 10.1145/3243734.3243860.

[17] Nick Doty. Mitigating browser fingerprinting in web specifications, 2019. URL https://www.w3.org/TR/fingerprinting-guidance/.

[18] Peter Eckersley. How unique is your web browser? volume 6205, pages 1–18, 01 2010. ISBN 978-3-642-14526-1. doi: 10.1007/978-3-642-14527-8_1.

[19] Steven Englehardt. Firefox 72 blocks third-party fingerprinting resources, Jan 2020. URL https://blog.mozilla.org/security/2020/01/07/firefox-72-fingerprinting/.

[20] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. pages 1388–1401, 10 2016. doi: 10.1145/2976749.2978313.

[21] Amin FaizKhademi, Mohammad Zulkernine, and Komminist Weldemariam. Fpguard: Detection and prevention of browser fingerprinting. pages 293–308, 07 2015. ISBN 978-3-319-20809-1. doi: 10.1007/978-3-319-20810-7_21.

[22] David Fifield and Serge Egelman. Fingerprinting web users through font metrics. volume 8975, pages 107–124, 01 2015. ISBN 978-3-662-47853-0. doi: 10.1007/978-3-662-47854-7_7.

[23] gk. Browser fingerprinting: An introduction and the challenges ahead, 9 2019. URL https://blog.torproject.org/browser-fingerprinting-introduction-and-challenges-ahead.

[24] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale. pages 309–318, 04 2018. ISBN 978-1-4503-5639-8. doi: 10.1145/3178876.3186097.

[25] U. Iqbal, S. Englehardt, and Z. Shafiq. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors. In *2021 2021 IEEE Symposium on Security and Privacy (SP)*, pages 283–301, Los Alamitos, CA, USA, may 2021. IEEE Computer Society. doi: 10.1109/SP40001.2021.00017. URL https://doi.ieeecomputersociety.org/10.1109/SP40001.2021.00017.

[26] Eiji Kitamura. Understanding "same-site" and "same-origin". URL https://web.dev/same-site-same-origin/.

[27] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Mitigating browser fingerprint tracking: Multi-level reconfiguration and diversification. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 98–108, 2015. doi: 10.1109/SEAMS.2015.18.

[28] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 878–894, 2016. doi: 10.1109/SP.2016.57.

[29] Pierre Laperdrix, Benoit Baudry, and Vikas Mishra. Fprandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In Eric Bodden, Mathias Payer, and Elias Athanasopoulos, editors, *Engineering Secure Software and Systems*, pages 97–114, Cham, 2017. Springer International Publishing. ISBN 978-3-319-62105-0.

[30] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. Browser fingerprinting: A survey. *ACM Trans. Web*, 14(2), April 2020. ISSN 1559-1131. doi: 10.1145/3386040. URL https://doi.org/10.1145/3386040.

[31] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. pages 541–555, 05 2013. ISBN 978-1-4673-6166-8. doi: 10.1109/SP.2013.43.

[32] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. Privaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 820–830, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee. ISBN 9781450334693. doi: 10.1145/2736277.2741090. URL https://doi.org/10.1145/2736277.2741090.

[33] Michael Schwarz, Florian Lackner, and Daniel Gruss. Javascript template attacks: Automatically inferring host information for targeted exploits. In *NDSS*, 01 2019. doi: 10.14722/ndss.2019.23155.

[34] Brave Software. Blockchain based digital advertising. *Whitepaper (10 February, 2021)< https://basicattentiontoken.org/BasicAttentionTokenWhitePaper-4.pdf> accessed*, 22, 2021.

[35] Naoki Takei, Takamichi Saito, Ko Takasu, and Tomotaka Yamada. Web browser fingerprinting using only cascading style sheets. In *2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*, pages 57–63, 2015. doi: 10.1109/BWCCA.2015.105.

[36] Martin Timko. Vylepšení rozšíření pro omezení volání JavaScriptu. Master's thesis, Brno University of Technology, Faculty of Information Technology, 2019. URL https://www.fit.vut.cz/study/thesis/21824/.

[37] Tom Van Goethem, Wouter Joosen, and Nick Nikiforakis. The clock is still ticking: Timing attacks in the modern web. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1382–1393, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450338325. doi: 10.1145/2810103.2813632. URL https://doi.org/10.1145/2810103.2813632.

[38] Zbyněk Červinka. Rozšíření pro webový prohlížeč zaměřené na ochranu soukromí. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2018. URL https://www.fit.vut.cz/study/thesis/21274/.

Príloha A

Výsledky testov

A.1 Am I unique testy

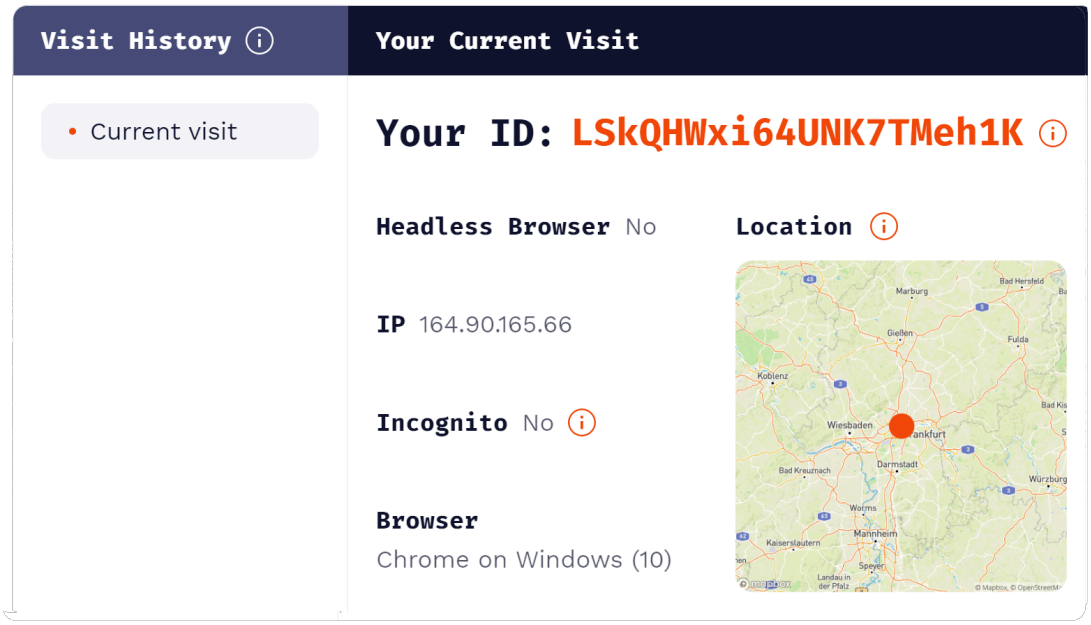
Hardware concurrency	0.36%	3
Java enabled	43.40%	false
Device memory	3.55%	4
List of plugins	Unique	Plugin 0: GjRIEix4; 48ePnTp0aNGDhwYMGjRIEChQoUq1aNmT; jx48. fPnzZs27dO. Plugin 1: JavaScript Portable Document Format Renderer; Portable Document Format; 4cOnTJEChQIEiRoUKFCBAAAgw4cOHDhw. Plugin 2: JavaScript document Viewer; ; DhQoUqVqVq16dOHjx4cOnTp06dOnz5cu. Plugin 3: s2btWrV; fvXr16dOnz5cOnTpUq169ePHDhQoUqV; kSJky5cu37dOHjx.
WebGL Vendor	0.07%	Google Inc. (Intel)
WebGL Renderer	<0.01%	ANGLE (Intel, Intel(R) HD Graphics 5500 Direct3D11 vs_5_0 ps_5_0, D3D11-20.19.15.5107)
WebGL Data	Unique	
Audio data	24.02%	Not supported
Media devices	Unique	videoinput audiooutput audioinput

Obr. A.1: Výsledok Brave na úrovni balanced na stránke Am I unique

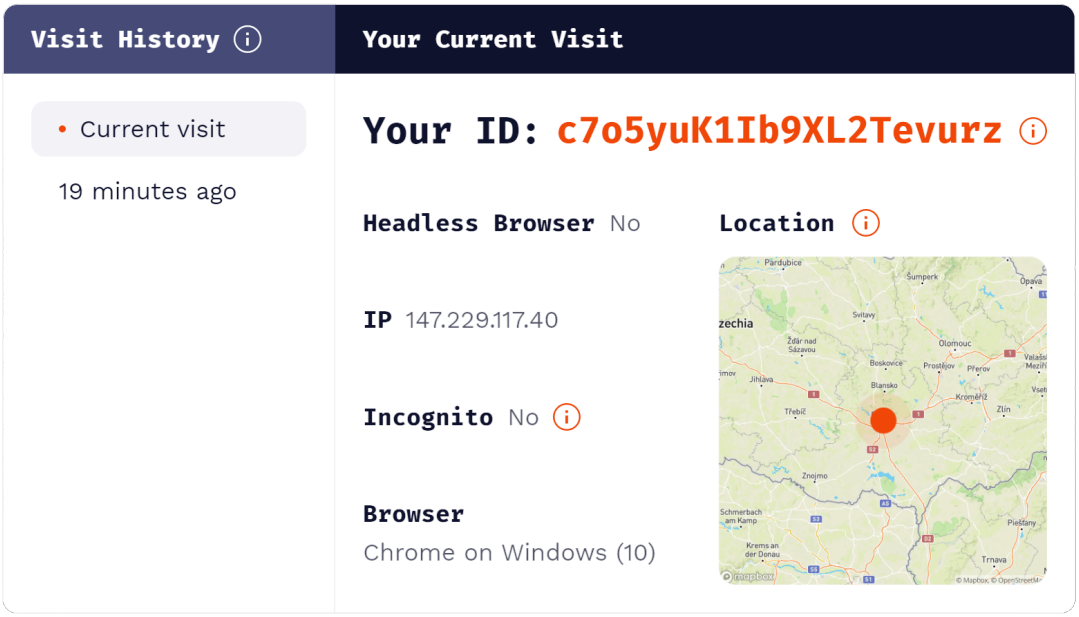
Hardware concurrency	9.83%	2
Java enabled	43.40%	false
Device memory	0.21%	1
List of plugins	Unique	Plugin 0: Native Client; ; HI19xlOe4aK6uQGcUHmjH5C9g0xzCGq4. Plugin 1: PDF plug-in; PDF ; 57lRWfcXvDnvo44AZiWGqeOZbGuR55vJ. Plugin 2: Iym3I27B; yd6J7ODiWwVSXZfnhk8-T6-PyZsi99j_; qtfsdG28bA-8vH2h. Plugin 3: GhuwOl9; 80LAcXG2TfJ6prgDh60aX4PJCVNXqXo; UFOVnyYVbyok_Ae. Plugin 4: Chrome document Display; doc ; YRQNvUOUN owWv22rB62E sVA4zFR56x.
WebGL Vendor	Unique	tzopsQ1n
WebGL Renderer	Unique	2 1ebGRP
WebGL Data	Unique	
Audio data	24.02%	Not supported
Media devices	Unique	videoinput audioinput videoinput audiooutput audioinput audioinput

Obr. A.2: Výsledok Chrome s JSR level 4 na stránke Am I unique

A.2 Fingerprint.js testy

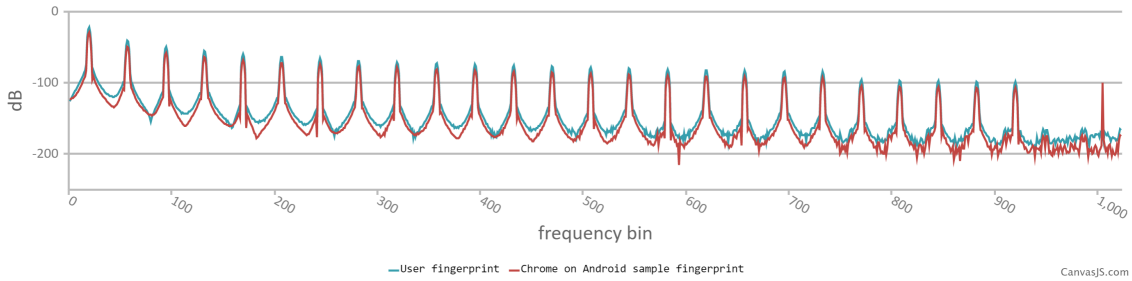


Obr. A.3: Výsledok Chrome s JSR level 4 na stránke fingerprintjs, použitie VPN a zmazané dáta

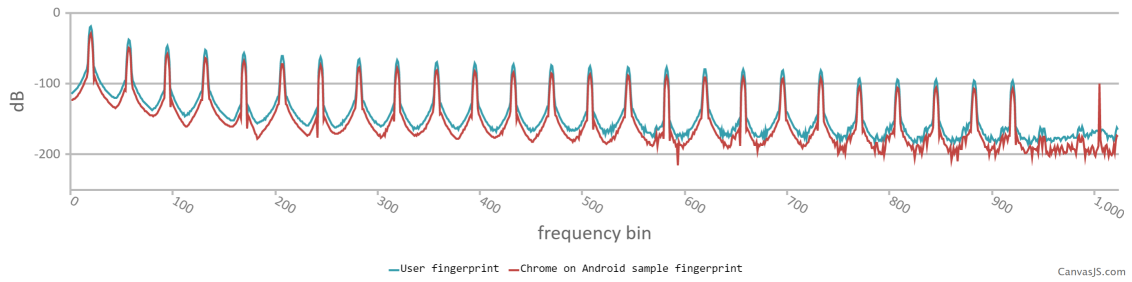


Obr. A.4: Výsledok Chrome s JSR level 4 na stránke fingerprintjs, nová relácia a nezmazané dáta

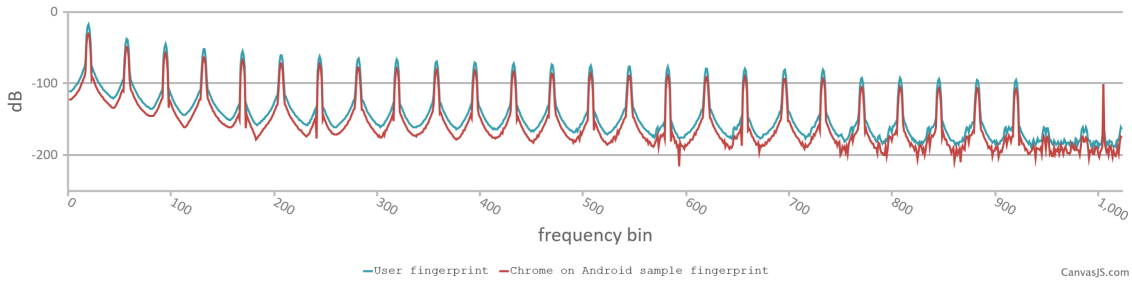
A.3 Web Audio testy



Obr. A.5: Výsledok Brave na úrovni balanced na stránke audiofingerprint.openwpm.com



Obr. A.6: Výsledok Chrome s JSR level 4 na stránke audiofingerprint.openwpm.com



Obr. A.7: Výsledok Firefox s JSR level 4 na stránke audiofingerprint.openwpm.com