VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

DETEKCE METOD ZJIŠŤUJÍCÍCH OTISK PROHLÍŽEČE

BROWSER FINGERPRINTING DETECTION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MAREK SALOŇ**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Saloň Marek, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Kybernetická bezpečnost
Název: **Detekce metod zjišťujících otisk prohlížeče**
Browser Fingerprinting Detection
Kategorie: Web
Zadání:

1. Seznamte se s nástroji získávajícími otisk prohlížeče, např. JSTemplate, FP-Scanner, FingerprintJS, Panopticlick, AmIUnique.org.
2. Nastudujte rozhraní WebExtensions používané webovými prohlížeči a seznamte se s projektem JavaScript Restrictor.
3. Navrhněte mechanismus detekující získávání otisku prohlížeče a zamezte přenosu dat na server jako rozšíření nástroje JavaScript Restrictor.
4. Návrh implementujte, pište dostatečně kvalitní a komentovaný kód, aby byl vedoucím práce do nástroje akceptován.
5. Implementaci otestujte.
6. Práci vyhodnoťte a navrhněte možná zlepšení.

Literatura:

- Pierre Laperdrix, Nataliia Bielova, Benoit Baudry a Gildas Avoine. 2020.Browser fingerprinting: A survey. ACM Trans. Web, řoč. 14, č. 2, str. 8:1-8:33.
- Antoine Vastel, Pierre Laperdrix, Walter Rudametkin a Romain Rouvoy. 2018. FP-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies. In Proceedings of the 27th USENIX Security Symposium. Baltimore, United States.
- Michael Schwarz, Florian Lackner a Daniel Gruss. 2019. JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits. In Network and Distributed Systems Security Symposium.
- Peter Snyder. 2018. Improving Web Privacy And Security with a Cost-Benefit Analysis of the Web API. Dizertační práce, University of Illinois at Chicago. Dostupné online https://www.peteresnyder.com/static/papers/improving-web-privacy-and-security-thesis.pdf

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

Hlavným cieľom tejto diplomovej práce je navrhnúť a implementovať mechanizmus, ktorý poskytuje ochranu proti bezstavovému sledovaniu odtlačkom prehliadača. Implementovaný nástroj má formu modulu, ktorý tvorí súčasť rozšírenia JavaScript Restrictor. Modul umožňuje špecifikovať heuristiky, podľa ktorých sa určuje, či navštívená stránka alebo nejaký jej prvok vykonáva extrakciu odtlačku prehliadača. V prípade detekcie podozrivej aktivity sú nasledujúce HTTP požiadavky stránky zablokované, aby sa zamedzilo odoslaniu získaného odtlačku na server. Implementácia a stanovené heuristiky boli otestované. Výsledný modul predstavuje účinný nástroj proti bezstavovému sledovaniu. Hlavným obmedzením implementácie je možné narušenie fungovania stránok blokovaním HTTP požiadaviek.

Abstract

The main goal of this thesis is to design and implement a mechanism that provides protection against stateless tracking with browser fingerprint. Implemented tool has a form of module that takes part of JavaScript Restrictor extension. The module allows to specify heuristics used for evaluation of visited sites that may contain browser fingerprint extraction. If suspicious activity is detected, all subsequent HTTP requests from that site are blocked to prevent the extracted fingerprint from being sent to the server. The implementation and defined heuristics were tested. The resulting module represents an effective tool against stateless tracking. The main limitation of the implementation is possible corruption of sites by blocking HTTP requests.

Kľúčové slová

bezstavové sledovanie, odtlačok prehliadača, detekcia, Web API, JavaScript, zber informácií, identifikácia, protiopatrenia, JavaScript Restrictor, rozšírenie prehliadača, internetové súkromie

Keywords

stateless tracking, browser fingerprint, detection, Web API, JavaScript, information gathering, identification, countermeasures, JavaScript Restrictor, browser extension, internet privacy

Citácia

SALOŇ, Marek. *Detekce metod zjišťujících otisk prohlížeče*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Libor Polčák, Ph.D.

Detekce metod zjišťujících otisk prohlížeče

Prehlásenie

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

.....
Marek Saloň
18. mája 2021

Poďakovanie

Veľmi rád by som sa poďakoval pánovi Ing. Liborovi Polčákovi, Ph.D za poskytnuté rady a nápady, ktoré som mohol využiť pri realizácií tejto práce.

Obsah

Kapitola 1

Úvod

Internetové sledovanie je technika zberu informácií o aktivite užívateľov internetu, ktorá má negatívny dopad na súkromie. Zbierané informácie môžu mať rôznorodý charakter, avšak v niektorých prípadoch sa môže jednať o osobné údaje, ktoré umožňujú užívateľa, či jeho zariadenie, jednoznačne identifikovať. Sledujúce strany priradia užívateľovi jedinečný identifikátor, vďaka ktorému môžu monitorovať jeho aktivitu a vytvárať si osobnostný profil. Tento profil môže byť natoľko konkrétny, že dokáže odhaliť skutočnú identitu človeka. Odstránením priradeného identifikátora bolo možné sledujúce strany výrazne obmedziť. To však podnietilo vznik nového mechanizmu sledovania, ktorý využíva identifikátor založený na vlastnostiach užívateľského zariadenia. Pri dnešnej rozmanitosti hardvérového a softvérového vybavenia týchto zariadení sa jedná o pomerne spoľahlivý spôsob identifikácie [21].

Táto práca sa venuje problematike internetového sledovania pomocou *odtlačku prehliadača*. Odtlačok prehliadača je forma bezstavového identifikátora, ktorý je založený na vlastnostiach zariadenia alebo internetového prehliadača. Narozdiel od stavových metód sledovania, napríklad pomocou cookies [9], takto vytvorený identifikátor nie je priamo uložený na strane klienta. Táto skutočnosť značne obmedzuje možnosti obrany proti identifikácií odtlačkom prehliadača a efektivita obranných mechanizmov je stále otázna [44]. Zameraním práce je analyzovať aktuálne metódy získavania odtlačku prehliadača a navrhnúť účinný mechanizmus obrany. Mechanizmus spočíva v pridaní funkcionality do rozšírenia *JavaScript Restrictor* [38], ktorá umožní detekciu procesu extrakcie odtlačku prehliadača a zamedzí prenosu tohto odtlačku na server.

Kapitola 2 popisuje štruktúru odtlačku prehliadača a jeho efektivitu z pohľadu použiteľnosti na účely identifikácie. Táto kapitola vychádza z citovaných štúdií, ktoré sa zaoberajú aktuálnym problémom narastajúcej popularity tohto typu sledovania. Koniec kapitoly 2 poukazuje na problémy jednotlivých prístupov obrany proti sledovaniu odtlačkom prehliadača. Kapitola 3 obsahuje technický popis princípu fungovania a implementácie rozšírení prehliadača. Táto kapitola ďalej predstavuje rozšírenie JavaScript Restrictor, ktoré slúži na zvýšenie súkromia a bezpečnosti prehliadača. Záver tejto kapitoly je venovaný návrhu metód detekcie získavania odtlačku prehliadača a zamedzeniu prenosu získaného odtlačku. Kapitola 4 popisuje implementáciu navrhnutej funkcionality v rámci projektu JavaScript Restrictor. Kapitola 5 poukazuje na výsledky testovania pridanej funkcionality a vyhodnocuje dosiahnuté výsledky. Záver kapitoly je venovaný popisu možných vylepšení.

Kapitola 2

Odtlačok prehliadača a jeho vlastnosti

Bezstavové sledovanie je pomerne nový mechanizmus, avšak jeho popularita každým rokom narastá. Toto je dôsledok viacerých opatrení proti stavovým mechanizmom, ktoré doteraz tvorili výraznú časť sledovacích techník. Z pohľadu bežného užívateľa to však nie je dobrá správa, pretože boj s pokročilými bezstavovými technikami je veľmi ťažký. Táto kapitola sa venuje detailnému popisu bezstavových metód sledovania, konkrétne sledovaniu pomocou odtlačku prehliadača. Sekcia 2.1 predstavuje základný koncept bezstavového sledovania odtlačkom prehliadača. Sekcia 2.2 obsahuje súhrn štúdií, ktoré sa venujú jedinečnosti a využitiu odtlačku prehliadača. Sekcia ďalej popisuje vlastnosti a štruktúru tradičného odtlačku prehliadača, ktorý je možné získať dostupnými nástrojmi určenými na zber odtlačkov. Popis jednotlivých zdrojov informácií a niektorých sofistikovanejších techník je obsiahnutý v sekcií 2.3. Záver kapitoly v sekcií 2.4 sa venuje možnostiam obrany proti identifikácií pomocou odtlačku prehliadača.

2.1 Bezstavové sledovanie

Bezstavové sledovanie je forma internetového sledovania, ktorá využíva vlastnosti klientskeho prostredia k vytvoreniu identifikátora. Tento identifikátor nie je nutné uchovávať na strane klienta, čím sa podstatne líši od stavového sledovania [10]. Identifikátor sa často skladá z kombinácie viacerých atribútov, ktoré sú špecifické pre daného používateľa alebo jeho zariadenie. Účinnosť tejto metódy je otázna, čo viedlo k mnohým výskumom [21], ktoré sa zoberajú jedinečnosťou bezstavového identifikátora.

Atribúty použité k vytvoreniu identifikátora sú softvérové a hardvérové vlastnosti klientskeho zariadenia a aplikácií. Od tohto sa potom odvíjajú pojmy ako *odtlačok zariadenia* alebo *odtlačok prehliadača*.

Odtlačok zariadenia je kombinácia informácií, ktoré sú špecifické pre dané zariadenie. Jedná sa o spojenie hardvérových parametrov, výpočtových schopností, softvérového prostredia a užívateľských nastavení zariadenia. Odtlačok prehliadača je konkrétnejší prípad odtlačku zariadenia. Tento typ odtlačku spolieha na identifikátor pozostávajúci z prehliadačom prístupných atribútov.

Najčastejšími atribútmi sú údaje získané z hlavičiek HTTP alebo údaje prístupné cez rôzne rozhrania jazyka JavaScript. To značne komplikuje aplikáciu obranných mechanizmov, pretože tieto atribúty sú úzko previazané s tým, ako internet aktuálne funguje. Ako

príklad je možné uviesť HTTP hlavičku `User-Agent`, ktorá slúži na získanie hlavných informácií o klientovom prehliadači. Vďaka tomu je server schopný upraviť správanie a obsah podľa prostredia, ktoré užívateľ aktuálne využíva. Odstránenie tejto hlavičky by tým pádom znamenalo mnohé problémy s kompatibilitou. Hlavička `User-Agent` je vhodným kandidátom na súčasť odtlačku prehliadača kvôli tomu, že ľudia využívajú rôzne prehliadače rôznych verzií na rôznych operačných systémoch. Tento atribút síce nemusí byť jednoznačne unikátny, avšak v spojení s ostatnými atribútmi tvorí silný základ na spoľahlivý identifikátor.

Odtlačok prehliadača v roli identifikátora predstavuje výraznú hrozbu z pohľadu súkromia. Jeho získanie je prakticky neviditeľné a nedá sa jednoducho zmeniť alebo odstrániť. To vzbudzuje obavy, pretože užívateľ nad tým nemá žiadnu moc. Peter Eckersley v štúdií [8] poukázal na niektoré využitia odtlačku prehliadača, ktoré majú negatívny vplyv na súkromie.

- **Odtlačok prehliadača ako globálny identifikátor.** Čím je zariadenie a jeho konfigurácia jedinečnejšia, tým je jedinečnejší aj odtlačok. V niektorých prípadoch môže byť odtlačok dostatočne jedinečný na to, aby sám o sebe slúžil ako identifikátor v globálnom meradle.
- **Odtlačok prehliadača s IP adresou ako mechanizmus na obnovu cookies.** Kombinácia odtlačku prehliadača so statickou IP adresou dokáže spoľahlivo poslúžiť na obnovu vymazaných cookies. Tento spôsob je možné pripočítať k mnohým existujúcim mechanizmom obnovy vymazaných cookies [5, 18].
- **Odtlačok prehliadača s IP adresou bez prítomnosti cookies.** Adresa IP je sama o sebe jedinečný identifikátor, avšak často pokrýva viacero koncových klientskych zariadení. Spojením s odtlačkom prehliadača vzniká mechanizmus, ktorý teoreticky umožní identifikovať jednotlivé zariadenia v lokálnej sieti. Zároveň nie je nutná prítomnosť cookies a iných mechanizmov, ktoré zanechávajú stopy na klientskom zariadení.

2.1.1 Všeobecné dôsledky sledovania

Sledovanie ako také predstavuje výrazné riziko z pohľadu súkromia. To však neznamená, že existujúce techniky sledovania sú používané len v neprospech užívateľov internetu. V praxi je možné rozdeliť používanie sledovacích techník do dvoch kategórií podľa zmyslu a samotného účelu [21].

- **Negatívne alebo deštruktívne používanie.** Tento spôsob zahrňuje tretie strany, ktoré sledujú užívateľa bez jeho vedomia alebo povolenia. To zahŕňa nepovolený zber osobných a identifikujúcich údajov o užívateľovi a jeho zariadení. Získané údaje je možné zneužiť napríklad na odhalenie slabín systému pre prípadný útok. Ďalší možný scenár predstavuje predaj údajov tretím stranám bez vedomia užívateľa.
- **Pozitívne alebo konštruktívne používanie.** Tretie strany získavajú potrebné údaje kvôli zlepšeniu služieb alebo ako bezpečnostné opatrenie. Užívatelia môžu byť varovaní v prípade zastaralého softvéru a nutnosti aktualizácií, ktoré zvyšujú bezpečnosť danej služby. Bezpečnosť internetových služieb môže byť posilnená aj mechanizmom verifikácie známeho zariadenia. Známe zariadenia majú jednoduchší prístup k službe, pričom každé nové zariadenie musí prejsť procesom verifikácie. Tento mechanizmus ochraňuje pred prístupom k službe zariadením neoprávnenej osoby.

2.2 Bezstavový identifikátor a jeho forma

2.2.1 Jedinečnosť odtlačku prehliadača

V roku 2009, Mayer vo svojej práci [22] ukázal, akým spôsobom je možné de-anonymizovať klientov pomocou JavaScript API. Pri jeho výskume si všimol, že prehliadač môže prejavovať zvláštnosti, ktoré sú podmienené operačným systémom. Mayer vo svojej práci popisuje aj experiment, kde využil niektoré atribúty objektov `window.navigator` a `window.screen` na identifikáciu užívateľov. Zo vzorky 1328 užívateľov bol schopný identifikovať až 96,23 %.

O rok neskôr, Peter Eckersley z *Electronic Frontier Foundation (EFF)* naviazal na túto tému experimentom *Panopticlick* [8]. Tento experiment zahŕňal prvý veľký prieskum spojený s využiteľnosťou odtlačku prehliadača. V rámci prieskumu sa podarilo nahromadiť takmer 470 161 odtlačkov, ktorých prevažná väčšina zahrňovala užívateľov dbajúcich na súkromie. Výsledky boli aj napriek tomu alarmujúce, pričom na základe HTTP hlavičiek a JavaScript API bolo možné jednoznačne identifikovať 83,6 % užívateľov. Podpora vtedajších pluginov Flash a Java toto číslo zdvihla až na 94,2 %. Experiment navyše skúmal aj stabilitu odtlačku prehliadača za nejaký čas. Až 65 % odtlačkov bolo možné znovu identifikovať pomocou predstaveného algoritmu.

Laperdix a kol. založili webovú stránku `amiunique.org` [20], ktorá bola súčasťou publikácie zverejnenej v roku 2016. Pomocou tejto stránky sa tímu podarilo získať 118 934 odtlačkov, z ktorých bolo 89,4 % jedinečných. Výskumný tím sa zameral na širšie spektrum atribútov prehliadača a zahrnul aj pokročilé metódy ako odtlačok pomocou Canvas API. Publikácia popisuje značný vývoj mechanizmov určených k získaniu odtlačku prehliadača. Predošlé prieskumy zaznamenali vysokú hodnotu entropie vďaka zoznamu pluginov a zoznamu systémových písiem. Popularita týchto atribútov výrazne klesla kvôli strate podpory pluginov, pretože predstavovali bezpečnostné hrozby. Na scénu prišli mechanizmy spojené s implementáciou HTML5 ako aktuálnym štandardom, ktorý priniesol nové aplikačné rozhrania jazyka JavaScript. Laperdix a kol. ako prví skúmali reálne využívanie Canvas API vo veľkom meradle, čím preukázali vysoký potenciál tohto mechanizmu. Tentokrát prieskum zahŕňal aj mobilné zariadenia (13 105 odtlačkov), ktorých odtlačok bol z 81 % jedinečný. Záverom preukázali, že HTTP hlavičky sú ďalší zdroj vysokej entropie, pričom použitím generickej hlavičky bez pluginov dokážu znížiť jedinečnosť odtlačku.

V roku 2018, Gómez-Boix a kol. v publikácií *Hiding in the Crowd* [16] zverejnili ďalší veľký prieskum týkajúci sa zberu a analýze odtlačkov prehliadača. Podarilo sa im získať až 2 067 942 odtlačkov z jednej z 15 najpopulárnejších francúzskych stránok. Hlavným cieľom prieskumu bolo získať odtlačky od obyčajných užívateľov, ktorí neprejavujú vysoký záujem o internetové súkromie. Narozdiel od predchádzajúcich štúdií, jedinečné odtlačky predstavovali iba 35,7 % z 1 816 776 odtlačkov pre počítače a 18,5 % z 251,166 odtlačkov pre mobilné zariadenia. Zásluhu nízkeho percentuálnemu podielu pripisujú reálnejšej vzorke účastníkov prieskumu, ako aj evolúcií na poli internetových technológií. Jedná sa hlavne o odstránenie podpory pluginov a celkovo vyššie povedomie o tejto problematike zo strany vývojárov. Publikácia ďalej priamo porovnáva všetky veľké výskumy v tejto oblasti na úrovni entropie jednotlivých atribútov odtlačku.

2.2.2 Využívanie odtlačku prehliadača v praxi

Panopticlick bol významný experiment, ktorý už pred desaťročím upozornil na riziko spojené s bezstavovým sledovaním. Odvtedy bolo realizovaných mnoho štúdií zameraných na

získanie štatistických údajov o adaptácií bezstavových mechanizmov sledovania v reálnom prostredí internetu.

- **Cookieless Monster** [32]
  V roku 2013, Nikiforakis a kol. preskúmali 10 000 najpopulárnejších stránok podľa rebríčka *Alexa*[1]. V rámci každej stránky navštívili do 20 rôznych odkazov. Prieskum bol zameraný na prítomnosť skriptov určených na extrakciu odtlačku prehliadača od spoločností BlueCava, Iovation a ThreatMetrix. Zistili, že 0,4 % stránok obsahuje skripty týchto spoločností.

- **FPDetective** [2]
  V roku 2013, Acar a kol. vykonali veľký prieskum zahrňujúci až 1 milión najpopulárnejších stránok. Na tento účel vytvorili nástroj *FPDetective*, ktorý pri prieskume stránok dokázal detektovať správanie podozrivé z extrakcie odtlačku prehliadača. Prieskum sa zameriaval hlavne na získanie zoznamu systémových písiem pomocou jazyka JavaScript a pluginu Flash. Celkovo navštívili 1 milión domovských stránok, kde na prvých 100 000 stránkach navštívili až 25 rozličných odkazov. Zistili, že 0,4 % z 1 milióna najpopulárnejších stránok obsahuje skript na extrakciu odtlačku vrátane skúmania systémových písiem. Pri prvých 10 000 domovských stránkach detektovali skúmanie systémových písiem pomocou Flash v 1,45 % prípadoch.

- **The Web Never Forgets** [1]
  V roku 2014, Acar a kol. zverejnili štúdiu vykonanú na 100 000 najpopulárnejších domovských stránkach. Štúdia sa venovala adaptácií pokročilých mechanizmov sledovania, konkrétne stavovým mechanizmom *evercookie* a *cookie syncing*. Okrem toho, štúdia skúmala aj prítomnosť získania odtlačku pomocou Canvas API. Zistili, že 5,5 % z navštívených stránok obsahuje skript, ktorý vykonáva extrakciu odtlačku vrátane použitia Canvas API. Pravdepodobnosť prítomnosti skriptov značne klesá pri menej populárnych stránkach.

- **A 1-million-site Measurement and Analysis** [10]
  V roku 2016, Engelhardt a Narayanan urobili prieskum pozostávajúci z detekcie mechanizmov stavového a bezstavového sledovania. Detekcia bezstavového sledovania bola realizovaná na 1 milióne najpopulárnejších domovských stránok. Na tento účel využili nástroj *OpenWPM*, ktorý umožnil automatizovanie prieskumu a detekciu pomocou inštrumentácie jazyka JavaScript. Prieskum bol zameraný na detekciu štyroch hlavných mechanizmov a výsledok bol nasledovný:

  - Vykresľovanie pomocou Canvas API - 1,4 %
  - Získanie zoznamu písiem pomocou Canvas API - 0,325 %
  - Odtlačok pomocou WebRTC API - 0,0715 %
  - Odtlačok pomocou AudioContext - 0,0067 %

- **Beyond Cookie Monster Amnesia** [3]
  V roku 2018, Al-Fannah a kol. vydali štúdiu, kde preskúmali 10 000 najpopulárnejších domovských stránok podľa rebríčka *Majestic*[2]. Narozdiel od predchádzajúcich štúdií, ktoré skúmali samotný jazyk JavaScript, táto štúdia monitoruje všetky údaje

[1] https://www.alexa.com/topsites
[2] https://majestic.com/reports/majestic-million

posielané z klientskeho zariadenia. Server získava odtlačok prehliadača práve vtedy, keď prehliadač odošle aspoň jeden z definovaných atribútov. Prieskum vo výsledku označil až 68,8 % stránok za stránky, ktoré zbierajú informácie vhodné pre zostavenie odtlačku prehliadača. Autori však prízvukujú, že nie všetky označené stránky musia vykonávať bezstavové sledovanie.

- **Fingerprinting the Fingerprinters** [17]
  V roku 2020, Iqbal a kol. predstavili nástroj *FP-Inspector*, ktorý využíva strojové učenie na detekciu extrakcie odtlačku prehliadača. Klasifikácia skriptov bola realizovaná kombináciou statickej a dynamickej analýzy jazyka JavaScript. Iqbal a kol. využili tento nástroj na prieskum 100 000 najpopulárnejších stránok. Zistili, že 10,18 % stránok vykonáva extrakciu odtlačku prehliadača. Pre vzorku tisíc najpopulárnejších stránok to bolo až 30.60 %.

2.2.3 Štruktúra odtlačku prehliadača

Odtlačok prehliadača často pozostáva z viacerých atribútov tak, aby ich kombinácia tvorila dostatočne jedinečnú hodnotu na následnú identifikáciu. Atribúty pozostávajú z informácií, ktoré reprezentujú daného užívateľa, jeho zariadenie, konfiguráciu alebo iné pozorovateľné charakteristiky.

Odtlačok prehliadača je možné rozdeliť do dvoch kategórií [7]. Tieto kategórie sú podmienené spôsobom prístupu k atribútom odtlačku.

- **Pasívny odtlačok prehliadača** je množina atribútov prístupných vo webových požiadavkách. Atribúty predstavujú metadáta internetovej komunikácie, ktoré sú zasielané spolu s požiadavkou a majú identifikačnú hodnotu. Získanie týchto atribútov nevyžaduje vykonávanie kódu na strane klienta. Hlavným zdrojom pasívneho odtlačku sú HTTP hlavičky a IP adresa.

- **Aktívny odtlačok prehliadača** je množina atribútov prístupných pomocou jazyka JavaScript alebo iného programovacieho jazyka. Odtlačok je potom považovaný za aktívny, ak bol získaný vykonaním určitého kódu v klientskom prehliadači. Hlavným zdrojom aktívneho odtlačku je objektový model prehliadača (BOM) a API prehliadača.

Tabuľka 2.1 prezentuje príklad reálneho odtlačku prehliadača získaného online nástrojom *Cover Your Tracks*[3]. Tento nástroj je priamym nástupcom projektu *Panopticlick*, za ktorým stojí organizácia *Electronic Frontier Foundation (EFF)*. Nástroj simuluje rozličné techniky sledovania a vyhodnocuje účinok obranných mechanizmov rozšírení a prehliadača samotného. Na extrakciu odtlačku je využitá open-source knižnica `fingerprintjs2`, čo je predchádzajúca verzia modernej knižnice `fingerprintjs` [12].

Na účely kvantifikácie množstva identifikujúcich údajov v jednotlivých atribútoch odtlačku sa používa *entropia*. Entropiu si je možné intuitívne predstaviť ako zovšeobecnenie množstva rôznych možností pre náhodnú premennú. Čím je entropia vyššia, tým je atribút jedinečnejší a obsahuje viac identifikujúcich údajov.

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

Atribút	Typ	Bity	Hodnota
User agent	P/A	4.62	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
Accept	P	12.76	text/html, */*;
Content encoding			gzip, deflate, br
Content language	P/A		sk-SK,sk;q=0.9,cs;q=0.8,en-US;q=0.7,en;q=0.6
Plugins	A	3.21	Plugin 0: Chrome PDF Plugin; Portable Document Format; internal-pdf-viewer;...
Timezone offset	A	2.07	-60
Timezone	A	10.01	Europe/Bratislava
Screen resolution	A	2.44	1920x1080x24
System fonts	A	3.85	Arial, Arial Black, Arial Narrow, Book Antiqua, Bookman Old Style, Calibri, Cambria...
Cookies	A	0.16	Yes
Supercookie test	A	1.52	DOM localStorage: Yes, DOM sessionStorage: Yes, IE userData: No, openDatabase: true, indexed db: true
Canvas FP hash	A	4.97	184c42ab72952c9ea0b2ef182038a365
WebGL FP hash	A	8.0	bed718f7ecfa4d36219ac2a183148b3a
WebGL vendor	A	7.02	Google Inc.
WebGL renderer			ANGLE (Intel(R) HD Graphics 630 Direct3D11 vs_5_0 ps_5_0)
DNT header	A	1.01	False
Platform	A	1.36	Win32
Touch support	A	5.02	Max touchpoints: 10; TouchEvent supported: false; onTouchStart supported: false
AdBlocker	A	2.97	True
AudioContext	A	2.87	124.04347527516074
HW concurrency	A	2.44	4
Device memory	A	2.16	8

Tabuľka 2.1: Príklad odtlačku prehliadača získaný pomocou nástroja *Cover Your Tracks*.

Entropia H diskrétnej náhodnej premennej X, kde X je hodnota z $\{x_1, ..., x_n\}$ a $P(X)$ je funkcia hustoty pravdepodobnosti, je definovaná ako

$$H(X) = -\sum_{i=1}^{n} P(x_i)\, log_b\, P(x_i), \tag{2.1}$$

kde $b = 2$. Táto hodnota parametru b umožňuje reprezentovať entropiu v bitoch. Každý bit entropie potom znižuje pravdepodobnosť výskytu udalosti o polovicu. Tento vzťah sa využíva vo viacerých štúdiách [16] na vyhodnotenie entropie jednotlivých atribútov odtlačku.

Tabuľka 2.1 zobrazujúca príklad odtlačku prehliadača má nasledovnú schému:

- Stĺpec *Atribút* obsahuje názvy atribútov odtlačku prehliadača. Jednotlivé názvy sú v anglickom jazyku, čím priamo odkazujú na originálne názvy týchto atribútov.

- Stĺpec *Typ* určuje typ atribútu odtlačku na základe spôsobu jeho získania. Pasívny atribút je označený písmenom P a aktívny atribút písmenom A.
- Stĺpec *Bity* obsahuje veľkosť entropie konkrétnej hodnoty atribútu odvodenú vzťahom

$$I(X = x_i) = -log_2\, p(x_i), \tag{2.2}$$

  kde I je entropia hodnoty atribútu x_i vyjadrená v bitoch a $p(x_i)$ je pravdepodobnosť výskytu danej hodnoty atribútu x_i.
- Stĺpec *Hodnota* obsahuje konkrétnu hodnotu atribútu extrahovanú z prehliadača.

Odtlačok prehliadača z tabuľky 2.1 je tvorený atribútmi, ktorých hodnoty majú rozličnú entropiu. Je vhodné pripomenúť, že sa jedná o entropiu konkrétnych hodnôt atribútov podľa vzťahu 2.2 a nie o entropiu atribútov podľa vzťahu 2.1. Na druhú stranu, odtlačok bol vytvorený v pomerne štandardnom prostredí a entropia hodnôt atribútov môže z časti odrážať približnú entropiu atribútov.

Z príkladu je možné vidieť, že hodnoty atribútov, ktoré sú z globálneho hľadiska jedinečnejšie, majú vyššiu entropiu. Napríklad hodnota atribútu *Timezone* obsahuje až 10 bitov informácie, pretože pravdepodobnosť výskytu tohto časového pásma je nízka. Naproti tomu, v prípade atribútu *Cookies* je entropia veľmi nízka, pretože pravdepodobnosť zapnutých cookies je pomerne vysoká.

Ďalším faktorom je *konzistentnosť odtlačku*, ktorá výrazne ovplyvňuje celkovú entropiu. Celková entropia je množstvo informácie obsiahnutej kombinovane vo všetkých atribútoch. Niektoré atribúty môžu byť na sebe závislé a obsahovať časť spoločnej informácie. Napríklad, ak pre atribút *Timezone* je zoznam jazykov atribútu *Content language* netradičný, odtlačok je o to jedinečnejší.

2.3 Metódy získavania odtlačku prehliadača

2.3.1 Hlavička HTTP

Hlavičky protokolu HTTP sú hlavným zdrojom informácií pasívneho odtlačku prehliadača. Ich prítomnosť je z hľadiska internetovej komunikácie takmer nevyhnutná. V prípade odtlačku sú centrom záujmu čiastočne identifikujúce údaje, ktoré majú vo vzájomnej kombinácií dostatočne vysokú hodnotu entropie na účely identifikácie.

- `User-Agent` je hlavička obsahujúca informácie o aplikácií, operačnom systéme, prehliadači a jeho verzií. Táto hlavička dokáže byť veľmi špecifická a obsahuje vysokú hodnotu entropie.
- `Accept` je hlavička obsahujúca informácie o podporovaných formátoch odpovede. Často závisí od prehliadača a časom sa veľmi nemení.
- `Content-encoding` je hlavička obsahujúca informácie o kompresných metódach podporovaných prehliadačom.
- `Content-language` je hlavička obsahujúca informácie o preferovaných jazykoch. Táto informácia môže mať vysokú entropiu v prípade exotickejších jazykov.
- `Upgrade-Insecure-Requests` obsahuje logickú hodnotu o preferencií šifrovania. Logické hodnoty majú nízku hodnotu entropie, pretože obsahujú len jeden informačný bit.

2.3.2 Vlastnosti prehliadača

Vlastnosti prehliadača tvoria atribúty odtlačku, ktoré skúmajú rôzne funkcie podporované prehliadačom. Moderné webové stránky spoliehajú na širokú podporu API, ktoré poskytujú bohatú funkcionalitu. Na základe toho, API môžu vo veľkom množstve obsahovať informácie o prehliadači a zariadení samotnom.

Odtlačok môže pozostávať aj zo zoznamu prístupných objektov v rámci koreňového objektu `window` alebo hociktorého iného prístupného objektu. Keďže *Browser Object Model (BOM)* nie je štandardizovaný, množina prístupných objektov, atribútov a rozhraní sa môže líšiť. Okrem toho, niektoré rozšírenia prehliadača môžu prepisovať natívne metódy jazyka JavaScript. Takéto nekonzistentnosti môžu viesť k vysokej entropií odtlačku [40].

Objekt `navigator`

Objekt `navigator` objektového modelu prehliadača obsahuje množstvo informácií, napríklad o prehliadači a iných parametroch zariadenia, ktoré nesú vysokú hodnotu entropie. Nasledovné objekty a atribúty sú prístupné cez objekt `navigator` a často tvoria súčasť odtlačku prehliadača.

- Atribúty `userAgent` a `language` obsahujú reťazce rovnakého typu ako ich ekvivalenty v hlavičkách HTTP.
- Atribút `platform` vracia reťazec reprezentujúci platformu, na ktorej beží prehliadač. Veľkosť entropie môže byť vysoká v prípade netradičného zariadenia. Príklad možných reťazcov: `'MacIntel'`, `'Win32'`, `'FreeBSD i386'`, `'WebTV OS'`.
- Atribút `productSub` pôvodne vracia číslo zostavy aktuálneho prehliadača. Jedná sa o zastaralý atribút, ktorý vracia konštantnú hodnotu podľa prehliadača. Chrome a Safari vracajú `20030107`. Firefox vracia `20100101`. Hodnota entropie je nízka, avšak využiteľné v kombinácií s inými atribútmi.
- Atribút `vendor` vracia reťazec reprezentujúci výrobcu aktuálneho prehliadača. Firefox vracia prázdny reťazec. Chrome vracia `'Google Inc.'` a Safari vracia `'Apple Computer, Inc.'`. Nízka hodnota entropie.
- Atribút `plugins` vracia pole objektov `Plugin`, ktoré reprezentujú jednotlivé pluginy prehliadača. Objekt `Plugin` obsahuje informácie o názve, popise a verzií pluginu. Zoznam pluginov s dodatočnými informáciami je využiteľný v rámci odtlačku prehliadača. Na druhej strane, pluginy sú už pomerne zastaralé a ich využitie je možné len v starších prehliadačoch. V prípade moderného prehliadača je entropia pomerne nízka.
- Atribút `connection` vracia rozhranie `NetworkInformation`, ktoré obsahuje informácie o sieťovom pripojení aktuálneho zariadenia. Rozhranie `NetworkInformation` poskytuje prístup k nasledujúcim atribútom:
  - Atribút `downlink` vracia efektívny odhad šírky pásma v megabitoch za sekundu (Mb/s).
  - Atribút `downlinkMax` vracia maximálnu rýchlosť sťahovania pre používanú technológiu v Mb/s.

  - Atribút `effectiveType` vracia typ pripojenia podľa technológie. Medzi možné hodnoty patria reťazce `'slow-2g'`, `'2g'`, `'3g'` alebo `'4g'`.
  - Atribút `rtt` vracia odhadovaný *round-trip time (RTT)* aktuálneho pripojenia.
  - Atribút `saveData` vracia logickú hodnotu o preferencií užívateľa šetriť dáta.
  - Atribút `type` vracia typ pripojenia zaradenia do sieťe. Medzi možné hodnoty patria napríklad reťazce `'bluetooth'`, `'ethernet'`, `'wifi'` a ďalšie [27].

  Kombinácia týchto informácií môže mať vysokú hodnotu entropie. Avšak, aktuálne sa jedná o experimentálne rozhranie, ktoré nemá podporu vo všetkých prehliadačoch.

- Atribúty `hardwareConcurrency` a `deviceMemory` obsahujú informácie o hardvérových parametroch zariadenia. Atribút `hardwareConcurrency` obsahuje počet logických jadier procesora. Atribút `deviceMemory` obsahuje približnú kapacitu operačnej pamäte v gigabajtoch. Tento atribút nie je podporovaný prehliadačmi Firefox a Safari. Nízka hodnota entropie, pokiaľ sa nejedná o neštandardné zariadenie.

- Atribút `cookieEnabled` vracia logickú hodnotu indikujúcu či sú cookies zapnuté, alebo nie.

- Atribút `doNotTrack` vracia logickú hodnotu o preferencií využívania hlavičky *Do Not Track (DNT)* [11].

- Atribút `keyboard` vracia rozhranie `Keyboard`, ktoré reprezentuje klávesnicu. Pomocou metódy `getLayoutMap()` je možné získať rozloženie klávesnice. Informácie o klávesnici sa môžu líšiť v závislosti od modelu a výrobcu. Tieto informácie môžu niesť nezanedbateľné množstvo entropie.

- Atribút `mediaDevices` vracia rozhranie poskytujúce prístup k pripojeným zariadeniam. Pomocou metódy `enumerateDevices()` je možné získať množstvo informácií o vstupných a výstupných zariadeniach dostupných v systéme.

Objekt `screen`

Objekt `screen` obsahuje informácie o aktuálnej obrazovke prehliadača. Informácie o obrazovke neobsahujú vysokú entropiu, ak sa nejedná o jedinečné zobrazovacie zariadenie. Rozlíšenie obrazovky môže byť nestále a jednoducho zmenené, čo značne znižuje stabilitu odtlačku. Nasledovné objekty a atribúty sú prístupné cez objekt `screen` a môžu tvoriť súčasť odtlačku prehliadača.

- Atribúty `height` a `width` vracajú výšku a šírku zobrazovacej plochy v pixeloch. Kombinácia hodnôt týchto atribútov vyjadruje rozlíšenie obrazovky zariadenia.

- Atribút `colorDepth` vracia hodnotu popisujúcu farebnú hĺbku obrazovky v bitoch.

- Atribúty `availHeight` a `availWidth` vracajú dostupnú výšku a šírku v okne prehliadača na zobrazenie obsahu.

API prehliadača

API prehliadača tvoria bohatý zdroj informácií a výrazne posilňujú celkovú entropiu odtlačku. Nasledovné rozhrania a objekty umožňujú prístup k informáciám, ktoré sú pomerne často využívané ako súčasť odtlačku prehliadača.

- Objekt `Date` reprezentuje konkrétny moment v čase v nezávislom formáte. Pomocou metódy `getTimezoneOffset()` je možné získať hodnotu posunu časového pásma oproti *Greenwich Mean Time (GMT)*.
- Rozhranie *Internationalization API* poskytuje pomocou objektu `Intl` funkcionalitu spojenú s formátovaním jazykovo citlivých údajov. Konštruktor `DateTimeFormat()` vytvára objekt `DateTimeFormat`, ktorý slúži na formátovanie údajov. Tento objekt obsahuje informácie o miestnych nastaveniach a možnostiach formátovania, ktoré sú prístupné metódou `resolvedOptions()`. Tieto informácie zahŕňajú reťazec reprezentujúci konkrétnu časovú zónu, typ kalendára, lokalizáciu a iné. Spomenuté informácie môžu byť z globálneho hľadiska veľmi konkrétne, pokiaľ sa jedná o neobvyklú časovú zónu alebo lokalizáciu.
- Rozhranie *Permissions API* poskytuje konzistentný spôsob prístupu k povoleniam, ktoré sa týkajú rozličných API v aktuálnom kontexte. Kombinácia povolení môže tvoriť súčasť odtlačku prehliadača. Príklady niektorých dotazovaných objektov:
  - `accelerometer`
  - `camera`
  - `geolocation`
  - `clipboard`
  - `magnetometer`
  - `microphone`
  - `notifications`
- Rozhrania *Audio/Video API* poskytujú HTML5 elementy `<audio>` a `<video>`, ktoré umožňujú prehrávať multimediálny obsah. Prehliadač podporuje vymedzené množstvo multimediálnych formátov, ktorých zoznam je možné získať pomocou spomínaných elementov. Objekty týchto elementov obsahujú metódu `canPlayType()`, ktorá vráti reťazec vyjadrujúci podporu konkrétneho formátu. Tým pádom je možné zavolať túto metódu na všetky známe formáty a získať podporované formáty zvukových súborov a videa. Zoznam formátov pravdepodobne nebude jedinečný, avšak môže obsahovať významné množstvo entropie.
- Objekt `AudioContext` reprezentuje graf vzájomne prepojených zvukových modulov `AudioNode`, ktoré umožňujú manipuláciu so zvukovými dátami. Atribút `destination` objektu `AudioContext` vracia objekt `AudioDestinationNode`, ktorý predstavuje posledný modul grafu, a teda výstupné zariadenie. Tento objekt obsahuje informácie o zvukovom výstupe a parametroch, ktoré môžu obsahovať určité množstvo entropie. Medzi tieto parametre patria:
  - `channelCount`
  - `channelCountMode`

  - `channelInterpretation`
  - `maxChannelCount`
  - `numberOfInputs`
  - `numberOfOutputs`

  Podobne je možné využiť objekt `AnalyserNode`, ktorý obsahuje nasledovné parametre potrebné pre frekvenčnú analýzu zvuku.

  - `fftSize`
  - `frequencyBinCount`
  - `minDecibels`
  - `maxDecibels`
  - `smoothingTimeConstant`

- Rozhrania *Web Storage API* a *IndexedDB API* poskytujú mechanizmy na ukladanie dát v prehliadači. Konkrétne sa jedná o objekty `localStorage`, `sessionStorage` a `indexedDB` prístupné cez koreňový objekt `window`. Prítomnosť týchto objektov tvorí atribút odtlačku prehliadača z tabuľky 2.1 pod názvom *Supercookie test*.

- Rozhranie *WebGL API* slúži na renderovanie interaktívnej 2D a 3D grafiky v elemente `<canvas>`. Toto API dokáže využiť hardvérovú akceleráciu poskytnutú grafickým adaptérom zariadenia. Objektu elementu `<canvas>` je možné priradiť *kontext WebGL* pomocou metódy `getContext('webgl')`, čo vráti `WebGL2RenderingContext`. Toto rozhranie poskytuje prístup ku konštantám popisujúcim rôzne parametre kontextu WebGL. Metódou `getParameter()` je možné získať hodnoty konštánt, medzi ktoré patria napríklad:

  - `VENDOR` (poskytovateľ WebGL a grafický adaptér)
  - `RENDERER` (rederovací engine a grafický ovládač)
  - `VERSION` (verzia WebGL)

  Tieto informácie môžu byť dostatočne podrobné na to, aby boli súčasťou odtlačku prehliadača a dosahovali vysokú entropiu. Najväčšie riziko predstavujú zariadenia s atypickými grafickými adaptérmi. *WebGL API* je pomerne komplexné rozhranie využívajúce hardvérové prostriedky, ktoré sa môžu v závislosti od zariadenia líšiť. Odtlačok môže tvoriť aj zoznam prístupných prostriedkov kontextu WebGL, ktoré závisia od schopností hardvéru.

2.3.3 Detekcia rozšírení prehliadača

Moderné prehliadače podporujú rozšírenia v podobe malých modulov rozširujúcich funkcionalitu prehliadača. Rozšírenia tvoria ďalší možný zdroj entropie, ktorý je možné využiť v rámci odtlačku prehliadača. Na druhej strane, rozšírenia nedefinujú konkrétne rozhrania pre prístup k informáciám tak, ako to je pri štandardných API. Z tohto dôvodu je nutné zvoliť alternatívne postupy [21] detekcie rozšírení a extrakcie dostupných informácií.

Sjösten a kol. v štúdií [41] skúmali využitie webových zdrojov na účely detekcie prítomnosti rozšírení prehliadača. Princíp spočíva v navštívení špecifickej adresy URL, ktorá

odkazuje na konkrétny zdroj rozšírenia. Existencia tohto zdroja potvrdzuje prítomnosť rozšírenia. Ako príklad je možné uviesť rozšírenie, ktoré obsahuje obrázok. Prehliadač pozná umiestnenie obrázka, ktorý je prístupný cez URL tvaru

`extension://<extensionID>/<pathToFile>`,

kde `extensionID` je identifikátor rozšírenia a `pathToFile` označuje cestu k obrázku. Tento spôsob detekcie nefunguje na všetky rozšírenia, pretože nie každé rozšírenie má takto prístupné zdroje. Sjösten a kol. boli schopný touto metódou detektovať 28 % z 43 429 rozšírení Chrome a 6,7 % z 14 896 rozšírení Firefox. Gulyas a kol. v štúdií [15] založenej na rovnakom princípe detekcie ukázali, že je možné identifikovať až 39.29 % z 7 643 užívateľov prehliadača Chrome detekciou 16 743 rozšírení.

Tradičnejším spôsobom detekcie rozšírení je identifikácia vedľajších efektov, ktoré tieto rozšírenia vyvolávajú. Vedľajšie efekty najčastejšie zahŕňajú modifikáciu DOM, kde rozšírenie pridá, odoberie alebo modifikuje elementy. Rozšírenia vyvolávajú vedľajšie efekty na základe dvoch podmienok.

- *Rozšírenia závisiace od lokality* vyvolávajú vedľajšie efekty podľa aktuálnej URL adresy. Ako príklad je možné uviesť rozšírenie sťahujúce videá z YouTube. Rozšírenie vytvára tlačidlo `Download` pod každým videom. Tlačidlo je možné detektovať v DOM pomocou skriptu prvej alebo tretej strany. Toto tlačidlo sa ale nikdy nezobrazí na inej lokalite ako YouTube, čo znemožňuje detekciu tohto rozšírenia na iných stránkach.

- *Rozšírenia závisiace na elementoch* vyvolávajú vedľajšie efekty v prítomnosti konkrétnych elementov HTML. Ako príklad je možné uviesť rozšírenie na správu hesiel, ktoré pri každom elemente `<form>` vytvorí príslušné tlačidlo na doplnenie údajov. Toto tlačidlo je možné detektovať každou stránkou, ktorá využíva element `<form>`. Za účelom detekcie týchto rozšírení stačí dočasne vytvoriť potrebný element a otestovať DOM na prípadné zmeny. Na tomto princípe funguje aj detekcia nástrojov na blokovanie reklám, ktorých prítomnosť je možné identifikovať vytvorením podozrivého elementu. Napríklad môže stačiť, keď element bude patriť do triedy, ktorá má v názve podreťazec `ads"`. Následná nedostupnosť alebo neviditeľnosť tohto elementu vyjadruje prítomnosť nástroja na blokovanie reklám.

Starov and Nikiforakis [43] poukázali na úspešnosť tejto metódy testovaním detekcie na 10 000 najpopulárnejších rozšíreniach prehliadača Chrome. Zistili, že dokážu detektovať 9 % rozšírení závisiacich na elementoch a 16,6 % rozšírení závisiacich na lokalite. Z 854 užívateľov boli schopný identifikovať 14,10 % len na základe detekcie rozšírení.

2.3.4 Algoritmické metódy

Algoritmické metódy zahrňujú atribúty odtlačku, ktoré nie je možné získať priamym prístupom. To znamená, že sa nejedná o hodnoty priamo prístupné cez rozhranie alebo objekt jazyka JavaScript. Tieto hodnoty tradične predstavujú výstup nejakého procesu, ktorý vracia výsledok na základe vstupu. Nasledujúce algoritmické metódy majú reálne nasadenie aj v praxi.

Enumerácia systémových písiem

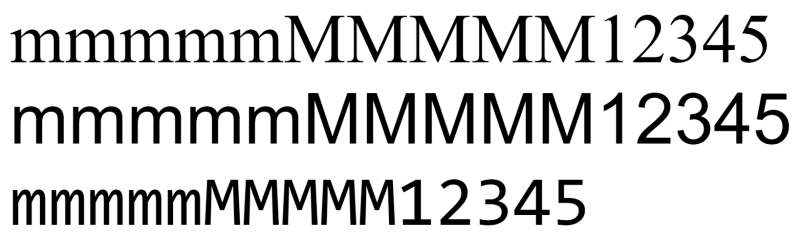
Zoznam systémových písiem môže slúžiť ako atribút odtlačku prehliadača. Tento zoznam bol už od počiatku považovaný za atribút s vysokou mierou entropie [8]. Napriek tomu,

že prehliadače nedisponujú možnosťou priamej enumerácie systémových písiem, existujú alternatívne spôsoby na dosiahnutie tohto cieľa.

Pôvodne bolo možné získať zoznam písiem cez plugin Flash, ktorý umožňoval priamy prístup. Takto získaný zoznam bol navyše nezoradený a stabilný, čo malo za následok zvýšenie entropie. Avšak, táto metóda je už irelevantná z pohľadu praktického využitia, pretože Flash už nie je podporovaný.

Nikiforakis a kol. v štúdií [32] objavili alternatívny spôsob enumerácie písiem pre prípad, že plugin Flash nie je aktívny. Tento spôsob poskytuje schopnosť zistiť dostupnosť konkrétneho písma jazykom JavaScript. Princíp je založený na vytvorení elementu `<div>` a vnoreného elementu `<span>`. Element `<span>` obsahuje preddefinovaný text spolu s preddefinovanou veľkosťou textu. Pomocou metód `offsetWidth` a `offsetHeight` elementu HTML je možné získať šírku a výšku elementu. Elementu je možné priradiť aj preferované písmo, avšak podpora písiem sa líši v závislosti od systému. Prehliadače majú definovanú množinu záložných písiem, ktoré sa použijú v prípade nedostupnosti zvoleného písma. Jednotlivé písma môžu mať odlišné rozmery pri nastavení rovnakej veľkosti písma. Obrázok 2.1 ilustruje túto skutočnosť na troch základným typoch písiem.

Zoznam podporovaných písiem je teda možné získať postupným porovnávaním rozmerov zvoleného písma s písmom záložným. Ak sú rozmery písma iné ako rozmery záložného písma, zvolené písmo je aplikované na element, a teda podporované. V opačnom prípade je vysoká pravdepodobnosť, že písmo nie je podporované a je použité záložné písmo. Pred samotným porovnávaním je nutné vytvoriť zoznam možných písiem, ktorého veľkosť ovplyvňuje časovú náročnosť tejto metódy. Nevýhoda tejto metódy spočíva v tom, že porovnávané písma sú pevne stanovené a nemusia zachytávať všetky podporované písma systému. Typicky sa využíva vysoká veľkosť preddefinovaného textu pre zohľadnenie čo najmenších odlišností. Niekedy môže nastať situácia, že porovnávané písma sú odlišné, ale majú rovnaké rozmery. Riešenie tejto situácie predstavuje využitie viacerých záložných písiem s rôznou šírkou.



Obr. 2.1: Príklad rozdielnych rozmerov písiem pri zachovaní rovnakej veľkosti textu. Obrázok obsahuje záložné písma rôznych rodín, ktoré sú zhora `serif`, `sans-serif` a `monospace`.

Canvas API umožňuje aplikovať podobný spôsob [10] enumerácie písiem v elemente `<canvas>`. Rozhranie `CanvasRenderingContext2D` obsahuje metódu `measureText()`, ktorá vracia objekt `TextMetrics`. Objekt `TextMetrics` reprezentuje rôzne metriky textu v elemente `<canvas>`, medzi ktoré patrí aj šírka textu. Podporu konkrétneho písma je možné detektovať priradením tohto písma do atribútu `font` a následným porovnaním šírky so záložnými písmami.

Canvas API

Canvas API predstavuje rozhranie poskytujúce nástroje na vykresľovanie 2D grafiky. Rozhranie využíva element `<canvas>`, ktorý reprezentuje plochu určenú na vykresľovanie. Vy-

kresľovanie je možné realizovať získaním kontextu metódou `getContext("2d")`. Získaný kontext `CanvasRenderingContext2D` obsahuje širokú ponuku metód, ktoré umožňujú kresliť grafické primitíva definovaním potrebných vlastností.

Výsledok vykresľovania rozhraním *Canvas API* môže slúžiť ako atribút odtlačku s vysokou entropiou a stabilitou. Prvýkrát túto metódu demonštrovali Mowery and Shacham v štúdií [30], ktorá odhalila, že vykresľovanie grafiky úzko súvisí na využívanom hardvéri, systéme a prehliadači. Vykresľovanie je navyše umiestnené mimo zobrazovaný obsah, a teda bežným okom nespozorovateľné. Zo vzorky 300 vykreslených obrázkov boli schopný získať až 50 rozličných výstupov.

Princíp je založený na vykreslení špecifických grafických primitív, ktoré čo najviac odzrkadlia prostredie prehliadača. Aj napriek tomu, že popis obrázku je rovnaký, rôzne prehliadače na rôznych systémoch môžu vracať rôzne výstupy. Odlišnosti tvoria zdroj entropie a je ich možné pozorovať na viacerých miestach.

- *Nekonzistentnosť systémových písiem* spôsobuje to, že rozličné systémy vykresľujú rovnaké písma rozdielne. Na získanie odtlačku sa využíva vykreslenie textového *pangramu*, čo je reťazec obsahujúci všetky písmena abecedy aspoň raz. Takto je možné zachytiť aj malú odlišnosť, napríklad v jedinom znaku abecedy. Obrázok odtlačkov 2.2 ilustruje využitie pangramov.
- *Metódy vyhladzovania hrán* sa môžu líšiť v závislosti na výrobcovi softvéru alebo hardvéru. Tieto odlišnosti môžu vytvárať rozdielne výstupy na úrovni pixelov.
- *Prelínanie a vypĺňanie* grafických objektov sú metódy, ktorých výstup sa môže líšiť podľa spôsobu renderovania daného zariadenia a prehliadača. Príklady odtlačkov na obrázku 2.2 zobrazujú vykresľovanie objektov, ktoré zahŕňajú tieto metódy.

Obr. 2.2: Príklad obrázkov, ktoré boli vykreslené za účelom získania odtlačku prehliadača. Tieto obrázky boli reálne využívané a nájdené na populárnych stránkach. Prevzaté z [10].

Získanie dát vykresleného obrázka je možné realizovať metódou `getImageData()`. Táto metóda vracia objekt `ImageData` reprezentujúci plochu elementu `<canvas>` vo forme pola

RGB hodnôt. Druhým spôsobom získania dát je využitie metódy `toDataURL(type)`, kde argument `type` definuje MIME typ cieľového formátu. Zo získaných dát je vhodné vytvoriť haš, aby nebolo nutné prenášať celé obrázky. Vlastnosti hašu umožňujú reflektovať všetky odlišnosti, avšak v omnoho kompaktnejšej forme.

WebGL API

WebGL API predstavuje rozhranie na renderovanie 2D a 3D grafiky, ktoré môže byť zneužité na vytvorenie odtlačku prehliadača. Rozhranie taktiež využíva element `<canvas>` a funguje podobne ako *Canvas API*. *WebGL API* poskytuje širšie možnosti vykresľovania a manipulácie s grafickými objektami. Výhodu predstavuje aj bližšia kooperácia s hardvérovými prostriedkami zariadenia, čo prináša zvýšenie rýchlosti.

Tieto schopnosti môžu byť zneužité podobným spôsobom ako v prípade *Canvas API*. Mowery and Shacham v tej istej štúdií [30] ukázali možné zneužitie *WebGL API* na identifikačné účely. Pomocou tohto rozhrania vytvorili 3D povrch, na ktorý aplikovali špeciálnu textúru. Textúra je tvorená štandardizovaným obrázkom, ktorý sa využíva na testovanie ostrosti a rozlíšenia šošoviek fotoaparátov. Následne pridali rôzne zdroje okolitého osvetlenia. Takto vytvoreným odtlačkom boli schopný získať až 50 rozličných výstupov z 270 testovaných zariadení. Odlišnosti vo výstupoch pripisujú rozdielnym prístupom zariadení k vykresľovaniu grafiky.

Cao a kol. [6] využili prostriedky *WebGL API* k vytvoreniu atribútu odtlačku prehliadača. Atribút bol vytvorený z výsledku 31 testov, ktoré boli zamerané na renderovanie 3D scén. Každá scéna sa zameriava na rozdielnu techniku grafického spracovania. V kombinácií s ostatnými atribútmi odtlačku boli schopný jednoznačne identifikovať 99 % z 1 903 testovaných zariadení.

WebRTC API

WebRTC API je rozhranie poskytujúce možnosť zachytávať, streamovať a posielať audio alebo video dáta. Hlavným konceptom je fungovanie na princípe *peer-to-peer*, čo odstraňuje nutnosť existencie prostredníka počas komunikácie.

Rozhranie sa používa napríklad na hlasovú komunikáciu v reálnom čase. Za účelom nájdenia najlepšej sieťovej cesty medzi účastníkmi komunikácie, každý účastník získa sieťové adresy dostupných zariadení z lokálnej siete a verejnej strany NAT. Tieto adresy sú poskytnuté webovej aplikácií bez nutnosti explicitného povolenia. To znamená, že aj napriek použitiu VPN, adresy užívateľov môžu byť prezradené [10].

Odtlačok prehliadača je možné doplniť o IP adresu zariadenia a posilniť celkovú entropiu odtlačku. Jednotlivé zariadenia lokálnej siete sú od verejnej siete abstrahované pomocou NAT, avšak *WebRTC API* umožňuje objaviť lokálne adresy týchto zariadení. Rozhranie `RTCPeerConnection` obsahuje metódy potrebné pre vytvorenie spojenia. Metóda `createDataChannel()` vytvorí fingovaný kanál. Metóda `createOffer()` potom vytvorí ponuku a metóda `setLocalDescription()` nastaví popis. Odosielateľ čaká kým druhá strana odpovie na ponuku. Po odpovedi je nutné dohodnúť vhodnú sieťovú cestu. Prijatie kandidátov vyvolá udalosť, na ktorú je možné reagovať obslužnou funkciou definovanou v atribúte `onicecandidate`. Adresy IP je možné prečítať z prijatých kandidátov.

AudioContext

Okrem toho, že z rozhrania `AudioContext` je možné získať veľké množstvo čiastočne identifikujúcich informácií, existujú aj algoritmické využitia tohto rozhrania. Z tohto hľadiska je princíp podobný ako pri ostatných algoritmických metódach, kedy softvérové a hardvérové odlišnosti zariadení vytvárajú rozdielne výstupy pre rovnaké vstupy.

Englehardt a kol. v štúdií [10] objavili nový spôsob vytvorenia atribútu odtlačku pomocou modulu `OscillatorNode`. Tento modul reprezentuje periodický priebeh, z ktorého je možné generovať zvukový signál. Zvukový signál je následne spracovaný v niekoľkých moduloch, pričom na výstup spracovania je aplikovaná hašovacia funkcia.

Prvý objavený spôsob využíva spracovanie modulom `AnalyserNode`, ktorým je možné extrahovať frekvenčné dáta signálu. Samotná extrakcia je zachytená obslužnou rutinou definovanou v atribúte `onaudioprocess` modulu `ScriptProcessorNode`. Zvukový signál je nakoniec utlmený modulom `GainNode` tesne pred samotným výstupom. Na frekvenčné dáta z modulu `AnalyserNode` je aplikovaná hašovacia funkcia, čo vytvára finálny atribút odtlačku.

Druhý objavený spôsob využíva modul `DynamicsCompressorNode` na spracovanie vygenerovaného zvukového signálu. Tento modul znižuje hlasitosť signálu na prevenciu pred skreslením. Výstup je uložený do modulu `OfflineAudioContext`. Uložené dáta sú rámec po rámci sčítané do jednej hodnoty, ktorá predstavuje kontrolný súčet. Tento súčet tvorí atribút odtlačku prehliadača.

Meranie času medzi udalosťami

Testovaním výpočtových schopností je možné odhaliť podstatné informácie o hardvéri zariadenia. V jazyku JavaScript je možné definovať sériu náročných výpočtov a určiť celkový výkon zariadenia na základe času, ktorý bol potrebný na vykonanie týchto výpočtov. Problém môže spočívať v nekonzistentnosti výsledku testov, pretože na výpočet vplýva mnoho vedľajších faktorov.

Testovať je možné rozličné komponenty zariadenia samostatne. Nakibly a kol [31] vytvárali atribút odtlačku pomocou zobrazovania komplexných 3D scén prostriedkami rozhrania *WebGL API*. Odtlačok pozostával z nameraného počtu snímok za sekundu, ktorý sa výrazne líšil v závislosti od zariadenia a použitého grafického čipu.

Sánchez-Rola a kol. [39] zisťovali rýchlosť procesora vykonávaním určitého počtu funkcií a meraním na to potrebného času. Zistili, že výsledky sú jedinečnejšie než tradičné odtlačky pomocou *WebGL API* alebo *Canvas API*.

Tento typ testovania vyžaduje presný spôsob merania času. V skriptoch zameraných na odber odtlačku prehliadača bolo nájdených [17] niekoľko použití rozhrania *Performance API*, ktoré umožňuje zaznamenávať časy udalostí s vysokou presnosťou. Medzi zachytávané udalosti patrí `domainLookupStart`, `domainLookupEnd`, `domInteractive` a `msFirstPaint`. Napríklad krátka doba vyhľadávania DNS môže prezradiť to, že nejaká stránka bola nedávno navštívená.

Podobne bolo nájdené animovanie elementov metódou `requestAnimationFrame()` objektu `window`. Metóda pri každom možnom prekreslení prehliadača volá príslušnú funkciu *callback*. Frekvencia volania tejto funkcie sa zhoduje s obnovovacou frekvenciou obrazovky, čím je možné vyjadriť počet snímok za sekundu. Tieto parametre môžu tvoriť atribút odtlačku.

2.4 Obrana proti identifikácií odtlačkom prehliadača

Odtlačok prehliadača môže byť použitý ako bezpečnostné opatrenie za účelom identifikácie oprávnenej osoby alebo zariadenia. Z pohľadu súkromia však predstavuje výrazné riziko. Odtlačok prehliadača môže byť zneužitý nasledujúcimi spôsobmi [7]:

- Identifikácia užívateľa na internete
- Korelácia online aktivity užívateľa medzi viacerými reláciami
- Sledovanie užívateľov bez kontroly alebo transparentnosti

Táto sekcia popisuje techniky [21] určené na potlačenie rizík spojených s odtlačkom prehliadača. Je nutné podotknúť, že tento problém aktuálne nemá jediné správne riešenie. Jednotlivé techniky zo sebou nesú aj plno obmedzení, ktoré je nutné akceptovať v prípade potreby lepšieho súkromia. Väčšina implementácií týchto techník preto spolieha na čo najlepšiu rovnováhu medzi súkromím a obmedzeniami.

2.4.1 Zvýšenie rozmanitosti zariadení

Hlavnú myšlienku tejto techniky predstavuje zvýšenie rozmanitosti odtlačkov prehliadača tak, aby sa reálne odtlačky stratili vo veľkom množstve rozdielnych odtlačkov. Táto myšlienka je podložená metódami tretích strán, ktoré sú schopné sledovať zariadenie medzi rozdielnymi internetovými stránkami prvých strán. Tretie strany v tomto prípade spoliehajú na *stabilitu* odtlačku, čo vyjadruje nemennosť odtlačku v závislosti od času. Tento cieľ je možné dosiahnuť používaním modifikovaného odtlačku prehliadača, napríklad vytvorením náhodného odtlačku alebo preddefinovaného falošného odtlačku. Takýto odtlačok môže predstavovať výraznú prekážku pre sledujúce tretie strany, pretože odtlačok je veľmi nestabilný. Táto technika má vysoký teoretický potenciál, avšak zavádza aj nežiadúci vedľajší efekt.

Porušenie stability náhodným odtlačkom je sprevádzané problémom možnej *nekonzistentnosti odtlačku*. Veľké množstvo verejne dostupných rozšírení prehliadača zameraných na súkromie využíva techniku modifikácie skutočného odtlačku prehliadača. Tieto modifikácie menia hodnoty atribútov prostredia prehliadača tak, aby sa z pohľadu odtlačku jednalo o iné zariadenie s inými vlastnosťami. Nikiforakis a kol. v štúdií [32] poukázali na nedostatky tohto prístupu. Rozšírenia často neponúkajú plné pokrytie atribútov a zavádzajú nekonzistentné zmeny. Hodnoty viacerých atribútov môžu byť na sebe závislé a zavedenie zmeny jedného atribútu môže spôsobiť nezmyselnosť hodnoty druhého atribútu. Náhodná zmena výhradne atribútu `navigator.userAgent` spôsobí nekonzistentnosť napríklad s hodnotou atribútu `appName` alebo HTTP hlavičkou `User-agent`. Zariadenie má tým pádom nereálnu konfiguráciu, ktorá sa za bežných okolností nevyskytuje. Tento fakt robí takéto zariadenie o to zraniteľnejšie voči identifikácií odtlačkom prehliadača.

Modifikácia hodnôt môže byť realizovaná viacerými spôsobmi. Čisto náhodný spôsob nepripadá v úvahu, pretože spôsobuje vysokú nekonzistentnosť, ak je odtlačok statický. *FP-Block*[4] využíva riešenie, kedy sa pre každú novú navštívenú doménu vytvorí nový odtlačok. Toto znemožňuje tretím stranám prepojiť identitu užívateľa medzi rozdielnymi stránkami. Na druhú stranu, nerozvážna zmena atribútov môže ovplyvniť použiteľnosť danej stránky. Server si môže mylne myslieť, že sa jedná o iný prehliadač, čo môže vyústiť v problémy

[4] https://satoss.uni.lu/software/fp-block/

s kompatibilitou. Využívanie reálnych odtlačkov iných zariadení je tiež možnosť, avšak nie je možné zakryť všetky ostatné vlastnosti skutočného zariadenia, ktoré nie sú obsiahnuté v odtlačku. Takéto zmeny atribútov je možné detektovať a zároveň odhadnúť skutočné hodnoty atribútov [44].

Algoritmické metódy vyžadujú rozdielny prístup a ich modifikovanie je náročnejšie. Jednoduché opatrenia v podobe rozšírení neumožňujú schovať všetky vlastnosti zariadenia a jeho hardvéru. Kvôli tomu sa využívajú techniky, ktoré modifikujú výstup týchto metód pridaním šumu. *Canvas Defender*[5] je rozšírenie pre Firefox zamerané na obranu proti identifikácií odtlačkom pomocou *Canvas API*. Prosté zamedzenie prístupu k rozhraniu *Canvas API* by mohlo spôsobiť nefunkčnosť potrebnej funkcionality hlavne vtedy, keď stránka využíva toto rozhranie poctivo. Prehliadač s rozšírením *Canvas Defender* vykresľuje obrázok v jeho pôvodnej podobe a šum je pridaný až pri snahe o extrakciu dát z obrázka. Podobné riešenie je možné použiť aj pri rozhraní *AudioContext API*, kde je možné zaviesť šum pomocou zvukového modulu.

Ďalším možným spôsobom je používanie viacerých prehliadačov. Toto riešenie predstavuje silné opatrenie proti stavovému aj bezstavovému sledovaniu. Aj keď odtlačky prehliadača sú dostatočne odlišné, stále je možné použiť odtlačok zariadenia.

2.4.2 Homogénny odtlačok prehliadača

Ďalšia technika, používaná na obranu proti identifikácií odtlačkom prehliadača, je založená na homogénnosti zariadenia a jeho odtlačku. Čisto teoreticky, ak sú všetky odtlačky rovnaké, nie je možné ich od seba navzájom rozoznať. Táto logika sa dá čiastočne aplikovať v praxi na zvýšenie pravdepodobnosti výskytu rovnakého odtlačku. Takýto odtlačok je menej jedinečný a obsahuje menej entropie, a tým pádom sťažuje identifikáciu.

Hlavným predstaviteľom tejto metódy je prehliadač *Tor*[6]. Tento prehliadač je postavený na využívaní *siete Tor*, ktorá poskytuje šifrovanie komunikácie a utajenie IP adresy. Samotná sieť však neposkytuje obranné mechanizmy proti typickému stavovému a bezstavovému sledovaniu. Obsah správ nie je zmenený, preto je možné sledovať užívateľov pomocou cookies alebo odtlačkom prehliadača. Prehliadač Tor vznikol aby vyplnil tieto medzery a poskytol komplexnú obranu proti obidvom typom sledovania. Z pohľadu opatrení týkajúcich sa odtlačku prehliadača, prehliadač Tor využíva koncept uniformity. To znamená, že prehliadač sa snaží vytvoriť generický odtlačok pre všetkých užívateľov. Medzi opatrenia patrí blokovanie alebo obmedzenie rozhraní, ktoré predstavujú riziko a často sa používajú pri vytváraní odtlačku. Ako príklad je možné spomenúť blokovanie `Canvas API`, používanie základnej množiny písiem, modifikácia atribútu `User-agent` v objekte `navigator` a hlavičke HTTP, a mnoho ďalších [34].

Zaručiť jednoznačnú homogénnosť nie je ľahká úloha. Samotný koncept predstavuje silnú ochranu proti odtlačku prehliadača, ale vyžaduje dodržiavanie striktných pravidiel. Tor vyžaduje používanie prehliadača v štandardnom rozlíšení `1000x1000` pixelov. V opačnom prípade, táto metrika môže slúžiť na detekciu konkrétneho užívateľa medzi ostatnými užívateľmi prehliadača Tor. Keďže typický odtlačok prehliadača Tor je pomerne jednoduché odlíšiť od ostatných odtlačkov, i najmenšie modifikácie štandardnej konfigurácie prehliadača môžu viesť k zvýšenej miere identifikovateľnosti.

Napriek všetkým implementovaným opatreniam existujú miesta, ktoré stále nesú určité množstvo entropie. Vlastnosti operačného systému a hardvéru vedia stále odkryť značné

[5] https://multilogin.com/canvas-defender/
[6] https://www.torproject.org/

množstvo informácií. Ako opatrenie proti metódam, ktoré testujú hardvér meraním času výpočtu, prehliadač znižuje presnosť rozhraní na meranie času. Článok [33] však ukazuje spôsob ako toto opatrenie obísť a identifikovať užívateľa prehliadača Tor pomocou rýchlosti procesora, rýchlosti pohybu myši a súradníc elementov HTML.

Prístup prehliadača Tor predstavuje efektívne riešenie z hľadiska ochrany proti sledovaniu odtlačkom prehliadača. Aj keď toto riešenie predstavuje menej kompromisov z pohľadu súkromia, použiteľnosť stránok môže byť obmedzená. Khattak a kol. [19] zistili, že 3,67 % z 1 000 najpopulárnejších stránok obmedzuje užívateľov prehliadača Tor blokovaním prístupu alebo degradovaním kvality ponúkaných služieb.

2.4.3 Zníženie rozsahu aplikačných rozhraní prehliadača

Ako už bolo naznačené, aplikačné rozhrania tvoria významný zdroj odtlačku prehliadača. Ich podpora je z časti nevyhnutná pre fungovanie moderného webu. Využitie rozhraní je neúmerne rozložené a niektoré API sa využívajú viac ako iné. Z pohľadu bezpečnosti a súkromia je rozloženie rizikových API tiež neúmerné. Snyder vo svojej práci [42] došiel k záveru, že niektoré API sú využívané vo veľmi nízkej miere, avšak predstavujú veľké bezpečnostné riziko. V kontraste k tomu, existujú často používané API, ktoré predstavujú nízke riziko. Ako príklad je možné uviesť rozhranie *WebGL API*, ktoré je používané na menej ako 1 % z 10 000 najpopulárnejších stránok. Ako už bolo ukázané, toto rozhranie je v značnej miere použiteľné na vytvorenie silného odtlačku prehliadača. Nutnosť explicitného povolenia API s vysokým rizikom a nízkou mierou využívania sa javí ako vhodné opatrenie.

Množstvo dostupných informácií, ktoré je možné získať pomocou skriptu, je možné obmedziť aj prísnejšími technikami. Extrémny príklad je celkové zakázanie spúšťania kódu jazyka JavaScript. Takéto opatrenie zamedzí serveru akýkoľvek zber informácií, ktoré tvoria aktívny odtlačok prehliadača. Sledujúci server je potom odkázaný čisto na pasívny odtlačok, prípadne stavové metódy sledovania. Problém spočíva v tom, že moderný web je z veľkej časti dynamický a funkcionalita prevažnej väčšiny webových stránok a aplikácií spolieha na JavaScript. Priamočiare zablokovanie skriptov vo veľkej miere obmedzí použiteľnosť väčšiny stránok a znemožní aj niektoré základné úkony ako napríklad prihlasovanie. Miernejším riešením je rozšírenie *NoScript*[7], ktoré umožňuje užívateľsky definovať kedy a kde blokovať JavaScript.

Globálne blokovanie z hľadiska použiteľnosti nepredstavuje ideálne riešenie. Existujú ale rozšírenia, ktoré využívajú databázu sledujúcich strán a všetky skripty od týchto strán automaticky blokujú. Medzi takéto rozšírenia je možné zaradiť *Ghostery*[8] a *Privacy Badger*[9].

[7] https://noscript.net/
[8] https://www.ghostery.com/
[9] https://privacybadger.org/

Kapitola 3

Zamedzenie sledovania v podobe rozšírenia prehliadača

Jedným zo spôsobov implementácie mechanizmov na ochranu pred identifikáciou odtlačkom prehliadača je vytvorenie rozšírenia, ktoré zvýši obranné mechanizmy prehliadača. Rozšírenie prehliadača disponuje potrebnými prostriedkami na detekciu neoprávneného zberu identifikujúcich informácií a zamedzeniu odoslania týchto informácií. V sekcií 3.1 popisujem technológiu WebExtensions API, ktorá slúži na vytváranie rozšírení prehliadača s vysokou kompatibilitou. Sekcia 3.2 predstavuje aktuálnu verziu rozšírenia JavaScript Restrictor, ktoré slúži na zvýšenie bezpečnosti a súkromia prehliadača. V sekcií 3.3 popisujem vlastný návrh mechanizmu, ktorý umožňuje detektovať získavanie odtlačku prehliadača na základe heuristík. Tento mechanizmus je navrhnutý za účelom následnej implementácie v rámci rozšírenia JavaScript Restrictor.

3.1 Rozšírenia prehliadača

Rozšírenia prehliadača sú mále programy, ktoré pridávajú funkcionalitu a upravujú správanie prehliadača. V roku 2015 bola pod konzorciom *W3C* vytvorená pracovná skupina, ktorá má za cieľ štandardizovať aplikačné rozhrania pre rozšírenia prehliadača [35]. Rozšírenia boli dovtedy vyvíjané na nezávislých technológiách, ktoré umožňovali kompatibilitu len s konkrétnymi prehliadačmi. Štandard mal priniesť jednotnú technológiu na vývoj rozšírení, ktoré budú natívne podporované všetkými hlavnými prehliadačmi. Hoci oficiálny štandard zatiaľ neexistuje, dominancia prehliadača Chrome prinútila ostatných výrobcov prispôsobiť sa.

Rozšírenia moderných prehliadačov sú založené na koncepcií pôvodných rozšírení prehliadača Chrome. Tieto rozšírenia sa spoliehajú na kombináciu esenciálnych webových technológií, a to *HTML*, *CSS* a *JavaScript*. Navyše majú prístup k rovnakým API prehliadača ako webové stránky, a aj k množine rozhraní špecificky určených pre použitie rozšíreniami. To znamená, že rozšírenia majú viac funkčných možností v rámci prehliadača ako webové stránky. Príklady využitia rozšírení prehliadača sú nasledovné [29]:

- **Vylepšenie alebo doplnenie webovej stránky** o funkcie, ktoré ponúknu dodatočné informácie alebo zlepšia použiteľnosť.
- **Personalizácia** internetového obsahu a prehliadača samotného, napríklad pomocou témy.

- **Pridanie alebo odobratie obsahu** umožňuje užívateľom blokovať nežiadúci obsah, formátovať formu internetových stránok, pridávať podporné elementy a mnoho ďalších.
- **Vytvorenie nástrojov a nových funkcií**, ktoré prehliadač normálne neposkytuje.
- **Pridanie vývojárskych nástrojov** umožňuje zefektívniť webový vývoj pre rôzne technológie.

3.1.1 Štruktúra rozšírenia WebExtensions API

Štruktúra rozšírení popísaná v tejto podsekcií odkazuje na rozhranie *WebExtensions API*[1], ktoré využíva prehliadač Firefox. Toto rozhranie vychádza zo záujmu vytvorenia spoločného štandardu a zahrňuje kompatibilitu s rozhraním *extension API*[2] používaným v prehliadači Chrome. Drobné rozdiely spomínaných rozhraní sú priblížené v podsekcií 3.1.3.

Rozhranie pozostáva z kolekcie súborov, ktoré sú zabalené na distribúciu a inštaláciu. Kolekcia súborov znázornená na obrázku 3.1 vychádza zo súboru `manifest.json`. Tento súbor predstavuje vstupný bod celého rozšírenia, pričom obsahuje informácie o celkovej štruktúre rozšírenia. Súbor `manifest.json` vďaka formátu *JSON* umožňuje deklaratívny zápis metadát potrebných na definíciu závislostí v rámci rozšírenia. Tento súbor je navyše jediný povinný súbor z kolekcie súborov znázornených na obrázku 3.1. Súbory sú zoskupené do komponent, ktoré zastávajú konkrétnu funkciu v rozšírení. Popis jednotlivých komponent sa nachádza nižšie.

Súbor `manifest`

Súbor `manifest` obsahuje trojicu povinných kľúčov, ktoré obsahujú potrebné informácie o rozšírení. Tieto kľúče sú zobrazené v kóde 3.1 spolu s možnými hodnotami. Kľúč `manifest_version` vyjadruje aktuálne používanú verziu tohto súboru. Kľúče `name` a `version` definujú meno, respektíve verziu rozšírenia.

```
"manifest_version": 2,
"name": "nazovRozsirenia",
"version": "1.0"
```

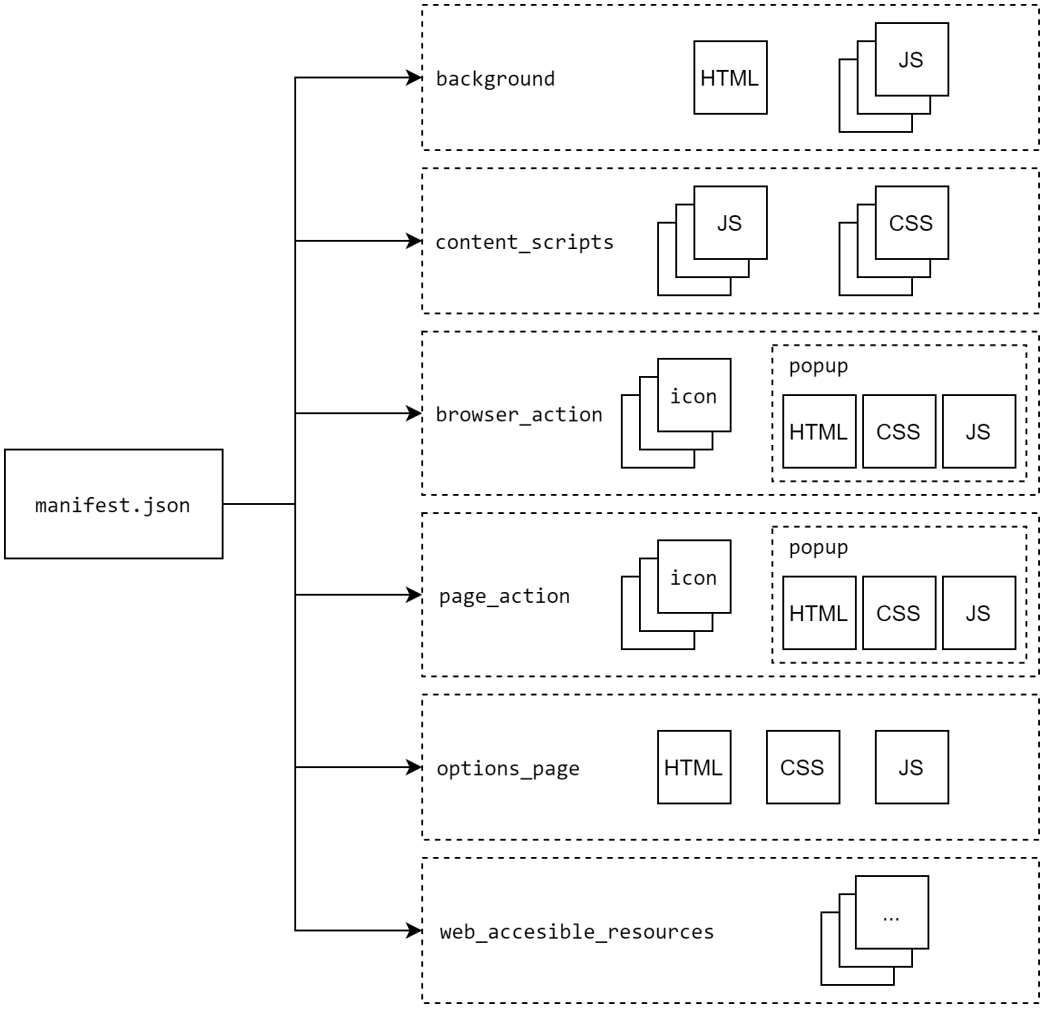
Kód 3.1: Príklad povinných kľúčov súboru `manifest`.

Ďalej nasledujú voliteľné kľúče, z ktorých niektoré patria medzi odporúčané. Medzi tieto kľúče je možné zaradiť napríklad `description` alebo `icons` znázornené kódom 3.2. Kľúč `description` obsahuje popis rozšírenia a kľúč `icons` obsahuje cestu k ikonám rozšírenia. Tieto ikony sú potom zobrazené v správcovi rozšírení prehliadača. Odporúča sa definovať ikony vo viacerých rozmeroch pre možnosť prispôsobenia vyšším rozlíšeniam obrazovky.

```
"description": "Popis rozsirenia",
"icons": {
  "48": "icons/logo-32.png"
  "96": "icons/logo-48.png"
}
```

Kód 3.2: Príklad odporúčaných kľúčov súboru `manifest`.

[1] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions
[2] https://developer.chrome.com/docs/extensions/



Obr. 3.1: Štruktúra komponent rozšírenia prehliadača založeného na WebExtensions API. Jednotlivé komponenty sú reprezentované príslušnými súbormi [23].

Komponenta `background`

Rozšírenia môžu vyžadovať uchovávanie stavu na nevyhnutne dlhú dobu od otvorenia prehliadača po jeho zavretie. Presne na tento účel sa využívajú *skripty na pozadí*, ktoré je možné definovať v súbore `manifest` pomocou kľúča `background`. Kód 3.3 zobrazuje príklad takej definície, kde definuje skript `background-script.js`, ktorý bude bežať od štartu samotného rozšírenia. Namiesto skriptu v jazyku JavaScript je možné definovať aj HTML dokument, ktorý umožňuje zahrnúť skripty a zároveň podporuje moduly štandardu *ECMA6*. Toto je možné realizovať zámenou kľúču `scripts` za kľúč `page` a definíciou zvoleného HTML dokumentu.

```
"background": {
  "scripts": ["background-script.js"]
}
```

Kód 3.3: Definícia skriptu na pozadí.

Skripty na pozadí bežia v osobitnom kontexte nezávislom na kontexte aktuálnej stránky. To znamená, že majú vlastný objekt `window`, DOM a môžu využívať všetky API prehliadača. Použitie rozhraní WebExtension je možné sprístupniť udelením povolení kľúčom `permissions`. Skripty na pozadí nemôžu priamo pristupovať k webovým stránkam, avšak môžu k tomu využiť komponentu `content_scripts`.

Komponenta `content_scripts`

Rozšírenie môže k stránke pristupovať pomocou komponenty `content_scripts`. Táto komponenta definuje *skripty s prístupom ku stránke*, ktoré sú načítané v rámci kontextu aktuálnej stránky. Skripty s prístupom ku stránke môžu manipulovať s DOM stránky, vytvárať asynchrónne požiadavky nezávisle na doménach a využívať podmnožinu rozhraní WebExtension.

Takéto skripty môžu byť priamo definované v súbore `manifest` kľúčom `content_scripts` alebo vložené pomocou skriptov na pozadí. Skripty s prístupom ku stránke môžu komunikovať so skriptami na pozadí a skriptami stránky pomocou správ. Namiesto skriptu JavaScript je možné do aktuálneho kontextu vložiť aj súbor štýlu CSS.

Komponenta `browser_actions`

Rozšírenie môže umiestniť tlačidlo na panel nástrojov prehliadača, a tým poskytnúť užívateľovi prístup k *užívateľskému rozhraniu rozšírenia (GUI)*. Príklad umiestnenia tlačidla rozšírenia na paneli nástrojov prehliadača je znázornený na obrázku 3.2.

Obr. 3.2: Tlačidlo rozšírenia, ktoré po stlačení zobrazí GUI prvok `popup`. Prevzaté z [13].

Takto vyzerajúce tlačidlo je možné zadefinovať v súbore `manifest` pomocou kľúča `browser_actions`. Príklad definície je znázornený kódom 3.4, pričom definícia obsahuje zanorené kľúče špecifikujúce ďalšie informácie.

```
"browser_action": {
  "default_icon": {
    "32": "images/icon-32.png"
  },
  "default_title": "Nazov",
  "default_popup": "popup.html"
}
```

Kód 3.4: Definícia tlačidla rozšírenia z panelu nástrojov prehliadača. Tlačidlo má definovanú ikonu, názov a HTML dokument reprezentujúci GUI.

Kľúč `default_icon` definuje ikonu tlačidla, kľúč `default_title` definuje názov a kľúč `default_popup` definuje GUI prvok `popup` s príslušným HTML dokumentom. Všetky zanorené kľúče v kóde 3.4 sú voliteľné.

Komponenta `page_action`

Tlačidlo je možné umiestniť aj priamo do adresného panelu pomocou komponenty, ktorá je definovateľná kľúčom `page_action`. Definícia ma rovnakú štruktúru ako v prípade tlačidla panelu nástrojov. Hlavný rozdiel spočíva v tom, že tlačidlo adresného panelu sa viaže na konkrétne stránky. Toto tlačidlo zároveň signalizuje aktivitu na danej stránke napríklad úpravou štýlu ikony.

Obr. 3.3: Tlačidlo rozšírenia umiestnené v adresnom paneli.

Komponenta `options_page`

Rozšírenie umožňuje špecifikovať stránku obsahujúcu nastavenia daného rozšírenia. Užívatelia tak majú prístup k dedikovanej stránke cez správcu rozšírení prehliadača.

Stránka je štandardne tvorená HTML dokumentom v kombinácií s CSS a skriptom v jazyku JavaScript. Skript bežiaci na tejto stránke môže využívať všetky rozhrania WebExtension, ak sú povolené v súbore `manifest`.

Kľúč `options_page` je už zastaralý a bol nahradený kľúčom `options_ui`, ktorý podporuje viaceré zanorené kľúče pre vyššiu flexibilitu. Príklad definície stránky nastavení je znázornený kódom 3.5.

```
"options_ui": {
  "page": "options/options.html"
}
```

Kód 3.5: Definícia stránky nastavení, ktorá sa nachádza v HTML dokumente definovanom zanoreným kľúčom `page`.

Komponenta `web_accesible_resource`

Komponenta s kľúčom `web_accesible_resource` definuje zdroje konkrétneho rozšírenia. Zdroje môžu byť tvorené rôznymi typmi súborov od obrázkov po textové dokumenty. Tieto zdroje sú prístupné danému rozšíreniu a môžu byť odkazované aj zo strany skriptov s prístupom ku stránkam a skriptov samotných stránok. Definícia v súbore `manifest` je znázornená kódom 3.6.

```
"web_accessible_resources": ["images/photo.png"]
```

Kód 3.6: Definícia zdroja, ktorý je reprezentovaný fotkou vo formáte PNG.

Prístup k vyššie definovanému zdroju je možné realizovať pomocou špeciálnej URL adresy tvaru

`extension://<extensionID>/images/photo.png"`,

kde `extensionID` po novom neobsahuje identifikátor rozšírenia, ale náhodne generovaný reťazec. Toto opatrenie je zavedené proti získavaniu odtlačku prehliadača rozšíreniami, čo však už bolo popísané v podsekcií 2.3.3.

Používateľské rozhranie rozšírenia

Rozšírenie prehliadača môže obsahovať viaceré prvky GUI, ktoré sa vytvárajú rovnakými spôsobmi ako webové stránky (HTML + CSS + JS). Medzi tieto prvky je možné zaradiť napríklad:

- Vyskakovacie okno (`popup`) je okno, ktoré je možné zobraziť kliknutím na tlačidlá z panelu nástrojov alebo adresného panelu. Okno zmizne kliknutím mimo daného okna.
- Bočná lišta (`sidebar`) má podobu vertikálneho panelu, ktorý sa zobrazí na boku od zobrazovaného webového obsahu.

Stránka rozhrania

Rozhranie môže obsahovať aj osobitné stránky, ktoré nie sú naviazané na žiadne prvky GUI. Tieto stránky sú tvorené štandardným spôsobom, avšak narozdiel od iných komponent, neobsahujú žiadny záznam v súbore `manifest`. Stránky rozhrania majú prístup k všetkým povoleným rozhraniam WebExtension podobne ako skripty na pozadí.

3.1.2 Rozhrania WebExtensions

Rozšírenia prehliadača majú prístup k rozhraniam WebExtensions, ktoré sú vyhradené pre použitie rozšíreniami. Použitie týchto rozhraní vyžaduje povolenie v súbore `manifest` definovaním hodnoty kľúča `permissions`. Rozhrania WebExtensions sú prístupné cez menný priestor `browser`. Nižšie sú priblížené niektoré z používaných rozhraní. Celý zoznam rozhraní WebExtensions spolu s kompatibilitou je dostupný na stránke MDN [24].

Storage API

Rozhranie *Storage API*[3] umožňuje rozšíreniam ukladať a načítať dáta z pamäte prehliadača. Zmeny uložených dát vyvolávajú udalosti, ktoré je možné zachytávať a patrične na ne reagovať. Rozhranie musí byť povolené, čo znamená, že kľúč `permissions` musí obsahovať reťazec `'storage'`.

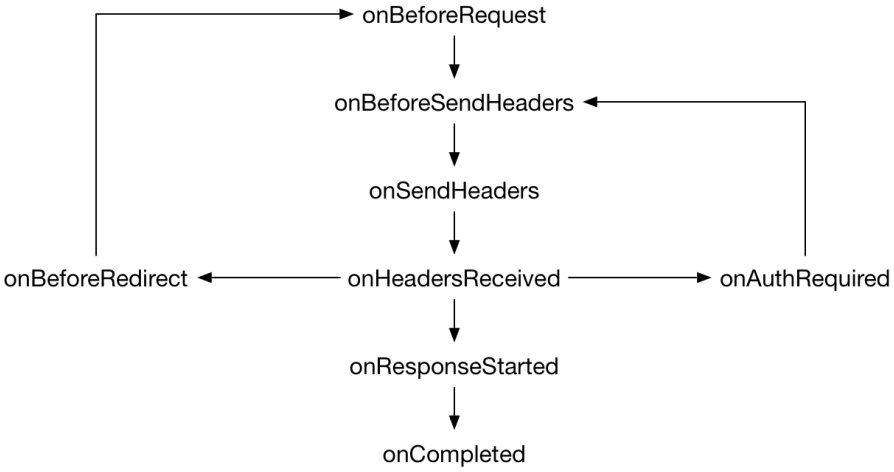
Rozhranie Storage API je založené na API prehliadača, konkrétne na *Web Storage API*. Narozdiel od API prehliadača, Storage API je asynchrónne rozhranie, ktoré vracia objekt `Promise`. Uložené dáta majú rozsah viazaný na rozšírenie, nie na konkrétne stránky. Dáta môžu byť ukladané v rôznych formátoch od klasických reťazcov až po objektové notácie, napríklad JSON. Jedným volaním je možné z úložiska získať viac hodnôt. Storage API definuje atribúty reprezentujúce jednotlivé typy úložiska.

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

- Atribút `storage.sync` reprezentuje *synchronizované úložisko*, ktoré sa synchronizuje na základe prihlásenia užívateľa v prehliadači. To znamená, že dáta z tohto úložiska sú prístupné užívateľovi na všetkých zariadeniach a inštanciách prehliadača, kde je tento užívateľ prihlásený.
- Atribút `storage.local` reprezentuje *lokálne úložisko*, pričom dáta uložené v tomto úložisku sú prístupne iba na konkrétnom lokálnom zariadení a prehliadači.
- Atribút `storage.managed` reprezentuje *spravované úložisko*, ktoré je spravované administrátorom domény alebo inou aplikáciou zariadenia. Z pohľadu rozšírenia sa jedná o úložisko iba na čítanie.

WebRequest API

Rozhranie *WebRequest API*[4] poskytuje prostriedky súvisiace so zachytávaním HTTP požiadaviek na úrovni rozšírenia. Toto rozhranie definuje sadu udalostí, ktoré je možné zachytiť. Zachytené udalosti obsahujú detailné informácie o odchádzajúcich a prichádzajúcich požiadavkách. Udalosti reprezentujú fázy spracovania HTTP požiadaviek prehliadačom, čo umožňuje reagovať v konkrétny moment. Postupnosť jednotlivých fáz spolu s názvami udalostí je zobrazená na obrázku 3.4. V prípade chyby počas spracovania HTTP požiadavky môže nastať udalosť `onErrorOccurred`.



Obr. 3.4: Udalosti reprezentujúce jednotlivé fázy spracovania HTTP požiadavku. Prevzaté z [28].

Reakciu na vyvolané udalosti je možné realizovať metódou `addListener()` objektu príslušnej udalosti. Táto metóda má 3 argumenty, kde prvý argument odkazuje na obslužnú funkciu, druhý argument obsahuje objekt `filter` a posledný argument obsahuje objekt `extraInfoSpec`. Objekt `filter` určuje, kedy má byť príslušná udalosť vyvolaná v závislosti od URL alebo typu požadovaného zdroja. Objekt `extraInfoSpec` je ako argument voliteľný a popisuje ďalšie inštrukcie spojené s udalosťou. Obslužná funkcia má na vstupe objekt `details`, ktorý reprezentuje HTTP správu. Vďaka tomu je možné pomocou *WebRequest API* vykonávať nasledujúce úkony:

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

- Získať prístup k hlavičkám a telu HTTP požiadavky a k hlavičkám HTTP odpovede.
- Zrušiť alebo presmerovať HTTP požiadavky.
- Modifikovať hlavičku HTTP požiadavky a odpovede.

3.1.3 Podpora rozšírení medzi prehliadačmi

Moderné prehliadače sa líšia vo viacerých aspektoch, medzi ktoré patrí aj podpora rozšírení. Prehliadače založené na projekte *Chromium* využívajú *extension API*[5], ktoré špecifikuje Google. Firefox s *WebExtension API*[6] podporuje aj rozšírenia *extension API*, avšak v opačnom prípade je nutné použiť *polyfill* [25]. Polyfill je kus kódu, ktorý doplňuje prehliadač o novú funkcionalitu za účelom kompatibility s inými alebo zastaralými prehliadačmi. To znamená, že rozšírenia postavené na *WebExtension API* potrebujú polyfill, aby mohli fungovať aj na prehliadačoch *Chromium*. Polyfill typicky vypĺňa medzery, ktoré sú spôsobené rozdielnymi implementáciami rozšírení. Hlavné principiálne rozdiely rozšírení sú nasledovné:

- **Menný priestor** sa líši v závislosti od prehliadača. Menný priestor umožňuje prístup k dostupným API. Prehliadač Firefox pristupuje k API cez objekt `browser`. Na druhej strane, prehliadače *Chromium* pristupujú k API cez objekt `chrome`.
- **Asynchrónne funkcie** sú implementované rozdielne podľa používaného prehliadača. Prehliadač Firefox implementuje API pomocou *Promises*. Naproti tomu, prehliadače *Chromium* implementujú API pomocou *spätných volaní (callback)*, kedy asynchrónna funkcia po skončení volá funkciu zo vstupu.
- **Štruktúra konfiguračného súboru** `manifest` nie je jednotná medzi prehliadačmi. Rozšírenia môžu vyžadovať vytvorenie osobitných súborov `manifest` pre jednotlivé prehliadače.
- **Podpora API** sa líši na základe prehliadača. Jednotlivý výrobcovia prehliadačov často predstavia nové API, ktoré môžu implementovať aj ostatné prehliadače. To však nie je pravidlo, a preto existuje niekoľko vzájomne nepodporovaných API.

Verzie súboru `manifest`

V roku 2018, Google predstavil aktualizáciu *extension API* a zároveň aj 3. verziu súboru `manifest`. Aktualizácia má za úlohu zlepšiť bezpečnosť, súkromie a výkon. Dosiahnutie týchto cieľov znamená nasledujúce hlavné zmeny [14]:

- *Service workers* nahradzuje stránku, ktorú bolo možné zadefinovať v rámci skriptov na pozadí. Nový mechanizmus je založený na *Web Workers API* a umožňuje beh vo vláknach na pozadí.
- *DeclarativeNetRequest API* je nové rozhranie poskytujúce deklaratívny prístup k spracovaniu HTTP správ. Toto rozhraniu umožňuje bezpečnejšie a rýchlejšie spracovanie založené na množine pravidiel. Pravidlá sú ukladané v súbore `manifest`, avšak umožňujú dynamický prístup.

[5] https://developer.chrome.com/docs/extensions/
[6] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions

- Vykonávanie *vzdialených skritpov* je zakázané. Rozšírenie môže spúšťať len tie skripty, ktoré sa nachádzajú v balíku rozšírenia.
- Pridaná podpora *Promises* ako mechanizmu asynchrónnych funkcií rozšírenia. Alternatívne je stále možné používať aj spätné volania.

Aktuálne sa stále používa 2. verzia súboru `manifest`, kde všetky rozšírenia Firefox sú kompatibilné s rozšíreniami Chrome. Na druhej strane, Google nedávno prešiel na plnú podporu 3. verzie súboru `manifest`, čím odporúča vývoj rozšírení v tejto verzií. Niektoré zmeny však nie sú spätne kompatibilné, čo môže značne ovplyvniť rozšírenia z funkčného hľadiska. Ako príklad je možné uviesť *WebRequest API*, ktoré je v novej verzií `manifest` zablokované. Spomínané *DeclarativeNetRequest API* síce supluje *WebRequest API*, avšak ani zďaleka neponúka takú úroveň funkčnosti ako procedurálny prístup *WebRequest API*.

3.2 JavaScript Restrictor

JavaScript Restrictor (JSR) [38] je rozšírenie prehliadača zamerané na zlepšenie bezpečnosti a súkromia. Rozšírenie JSR je založené na WebExtensions API a podporuje širokú paletu prehliadačov, medzi ktoré patrí Firefox, Chrome, Opera a ostatné prehliadače založené na projekte Chromium.

Úlohou rozšírenia JSR je zlepšiť bezpečnosť a súkromie prehliadania internetu pomocou blokovania, obmedzenia alebo modifikácie API prehliadača. Ako už bolo ukázané v kapitole 2, API prehliadača predstavujú hlavný zdroj čiastočne alebo úplne identifikujúcich informácií.

Motivácia je založená na skutočnosti, že stránky využívajúce JavaScript majú takmer neobmedzený prístup k väčšine API. Niektoré API z bezpečnostných dôvodov vyžadujú explicitné povolenie užívateľa (napríklad Geolocation API), avšak väčšina z nich je voľne prístupná. Tým pádom, užívateľ prehliadača nemá prehľad o využívaní týchto API a ani prostriedky k ich kontrole.

Rozšírenie JSR predstavuje nástroj, ktorý užívateľovi umožní kontrolovať vybrané rozhrania, objekty a atribúty jazyka JavaScript. Z implementačného hľadiska sa jedná o obalenie konštrukcií jazyka JavaScript tak, aby k nim nebol umožnený priamy prístup [45]. To znamená, že sa vytvorí *obálka*, ktorá využíva lokálny priestor platnosti premenných. K API je potom nutné pristupovať cez *obálku*, v ktorej je možné výsledok volania ľubovoľne spracovať. Rozšírenie JSR týmto spôsobom podporuje nasledovné obmedzenia API prehliadača:

- *Network boundary shield (NBS)* je mechanizmus obaľujúci *WebRequest API* na prevenciu proti útokom, kde je prehliadač zneužitý ako proxy medzi lokálnou a verejnou sieťou.
- Obalenie `window.Date`, `window.performance.now()` a `window.PerformanceEntry` za účelom zníženia presnosti. Prostriedky poskytujúce vysokú presnosť merania času môžu byť zneužité na získanie odtlačku zariadenia alebo iné útoky spojené s časovou analýzou.
- Modifikácia *Canvas API*, kde metódy `canvas.toDataURL()`, `canvas.toBlob()` a `CanvasRenderingContext2D.getImageData()` vracajú dáta bieleho obrázku. Vďaka tomu je možné zabrániť získaniu odtlačku prehliadača pomocou *Canvas API*.

- Modifikácia atribútov `navigator.deviceMemory` a `navigator.hardwareConcurrency`, ktoré poskytujú informácie o hardvéri zariadenia a môžu byť zneužité na identifikáciu zariadenia.
- Monitorovanie *XMLHttpRequest (XHR)*, ktoré umožňuje asynchrónnu komunikáciu zo serverom. Server takto môže prenášať získané identifikačné údaje.
- Obalenie množiny objektov `ArrayBuffer`, ktoré môžu byť zneužité pri útokoch využívajúcich hardvér zariadenia.
- Obalenie objektov `window.SharedArrayBuffer` a `window.Worker`, ktoré je možné zneužiť na útoky časovou analýzou.
- Modifikácia lokalizačnej a časovej presnosti *Geolocation API* pre prípad, že užívateľ nechce odhaliť svoju skutočnú polohu.
- Pravidelné mazanie atribútu `window.name`, ktorý inak môže byť zneužitý na sledovanie v rámci karty prehliadača.

Niektoré obmedzenia môžu znamenať problémy s kompatibilitou a stabilitou internetových stránok. Cieľom rozšírenia JSR je nájsť správny kompromis medzi bezpečnosťou a funkcionalitou. Z tohto dôvodu boli zavedené nasledovné úrovne ochrany, ktoré je možné nastaviť cez GUI rozšírenia. Detailný popis úrovní je dostupný na stránke JSR [37]. Mimo pevne stanovené úrovne je možné špecifikovať aj vlastné úrovne ochrany.

- **Úroveň 0** - celkové vypnutie ochranných mechanizmov rozšírenia JSR.
- **Úroveň 1** - minimálna úroveň ochrany, ktorá by nemala výrazne ovplyvniť funkčnosť webových stránok.
- **Úroveň 2** - základná úroveň ochrany, ktorá poskytuje dobrý pomer medzi ochranou a funkčnosťou stránok.
- **Úroveň 3** - vysoká úroveň ochrany, kedy sú všetky obranné mechanizmy aktívne.

3.3 Návrh opatrení zamedzujúcich bezstavové sledovanie

3.3.1 Problematika detekcie získavania odtlačku prehliadača

Získavanie informácií o prehliadači pomocou jazyka JavaScript je pomerne priamočiare. Skripty na strane klienta majú prístup k širokému spektru prostriedkov, ktoré umožňujú extrakciu informácií použiteľných na budovanie odtlačku prehliadača. Niektoré sofistikovanejšie metódy poskytujú prístup k informáciám, ktoré nie sú priamo prístupné. Avšak aj v tomto prípade platí, že sa využívajú štandardne prístupné prostriedky jazyka JavaScript, ktoré sú niekedy nevyhnutné na správne fungovanie webu.

Detekcia zneužitia prostriedkov jazyka JavaScript na extrakciu odtlačku prehliadača je komplexný problém. Nie vždy je jasné či skript vykonáva korektnú činnosť, alebo vytvára atribúty odtlačku. Prístup k *Canvas API* môže znamenať vykreslenie 2D grafiky na navštívenej stránke, ale aj to, že navštívená stránka vykonáva extrakciu odtlačku prehliadača. Hodnota atribútu `userAgent` je často využívaná na určenie prostredia prehliadača, aby stránka mohla poskytnúť kompatibilný obsah. Na druhej strane, tento atribút obsahuje pomerne vysokú entropiu a často tvorí súčasť odtlačku prehliadača. Laperdrix a kol.

v článku [21] poukázali na znaky, ktoré môžu indikovať, že skript vykonáva aktivitu spojenú so získavaním odtlačku prehliadača.

- **Prístup ku špecifickým funkciám prehliadača**, ktoré obsahujú identifikujúce informácie. Výskum [17] v tejto oblasti zaznamenal množstvo zneužívaných objektov a metód, ktorých použitie je časté v skriptoch extrahujúcich odtlačok prehliadača.
- **Získavanie veľkého množstva identifikujúcich informácií** môže znamenať pokus o vytvorenie odtlačku prehliadača. Samotný prístup k atribútu `userAgent` vo všeobecnosti nemá dostatok entropie, aby predstavoval spoľahlivý identifikátor. V prípade, že sa jedná o sériu viacerých prístupov k väčšiemu množstvu API, pravdepodobnosť získavania odtlačku sa zvyšuje.
- **Viacnásobný prístup k rovnakému prostriedku** môže znamenať, že sa jedná o niektorú zo sofistikovanejších metód spomenutých v podsekcií 2.3.4. Napríklad viacnásobné volanie *Canvas API* s vysokým počtom rôznych hodnôt atribútu `font` môže znamenať získavanie zoznamu písiem.
- **Hašovanie hodnôt** sa vďaka vlastnostiam hašovacích funkcií môže vyskytovať pri extrakcií odtlačku prehliadača. Server môže hašovať celý odtlačok, aby ušetril dáta a urýchlil samotný prenos.
- **Posielanie informácií na vzdialenú adresu**, ktoré obsahuje veľké množstvo identifikujúcich dát, môže znamenať prenos odtlačku prehliadača.
- **Obfuskácia (obfuscation) zdrojového kódu** je technika zahmlievania skutočného významu a princípu fungovania kódu. Táto technika sa niekedy využíva na ochranu intelektuálneho vlastníctva zdrojového kódu, avšak môže slúžiť aj na zakrytie skriptu extrahujúceho odtlačok prehliadača.

3.3.2 Heuristiky detekcie získavania odtlačku prehliadača

Návrh detekcie získavania odtlačku prehliadača vychádza z prostriedkov prístupných pomocou WebExtensions API. Detekcia je založená na vybraných heuristikách, ktorých starostlivé nastavenie umožňuje dosiahnuť vysokú účinnosť a presnosť. Rovnaký postup bol použitý v štúdiách [16, 10] a ukázalo sa, že je možné dosiahnuť nízku úroveň falošne pozitívnych nálezov. To znamená, že je nízka pravdepodobnosť označenia nesledujúcej stránky za stránku sledujúcu. Takto je možné vyhnúť sa nežiadúcim vedľajším efektom protiopatrení na nesledujúcich stránkach a zlepšiť celkovú použiteľnosť.

Z pohľadu rozšírenia prehliadača je možné aplikovať vybrané heuristiky na rôzne časti webovej komunikácie. Keďže odtlačok prehliadača je zo značnej miery tvorený informáciami dostupnými cez prostredie jazyka JavaScript, aplikácia heuristík na prostriedky tohto jazyka predstavuje použiteľný spôsob detekcie sledovania. Rozšírenie prehliadača umožňuje vložiť skript do aktuálneho kontextu stránky ešte pred jej samotným spracovaním. Týmto spôsobom je možné inštrumentovať prostriedky jazyka JavaScript, najmä API prehliadača a objekty BOM, ktoré sú často zneužívané na vytváranie odtlačku. Inštrumentácia týchto prostriedkov poskytuje schopnosť zaznamenávať jednotlivé prístupy, a na základe toho ich vyhodnocovať zvolenými heuristikami.

Heuristiky sú postavené na informáciách o prístupoch k prostriedkom jazyka JavaScript. Tento spôsob umožňuje abstrahovať vykonávaný kód v rámci webovej stránky do jedného

celku, na ktorý sú potom aplikované heuristiky. Princíp zaznamenávania prístupov je blízky dynamickej analýze kódu, vďaka čomu umožňuje zachytiť behaviorálne črty bežiaceho kódu bez nutnosti statickej analýzy. Statická analýza je v tomto prípade vynechaná, pretože detekcia na základe štruktúry kódu môže byť výrazne ovplyvnená obfuskáciou kódu.

Inštrumentáciou kódu je možné zaznamenať, ktoré skripty jednotlivých domén pristupujú k citlivým prostriedkom jazyka JavaScript. Z tohto dôvodu je nutné vybrať množinu prostriedkov, ktoré nesú nezanedbateľnú hodnotu entropie. Prístup k citlivým prostriedkom je nutné monitorovať a jednotlivé prístupy zaznamenávať. Vstup pre heuristiky je tvorený množinou záznamov, ktoré obsahujú informácie o prístupoch k monitorovaným prostriedkom.

Aplikácia heuristík detektujúcich extrakciu odtlačku prehliadača

Monitorované prostriedky jazyka JavaScript predstavujú objekty a metódy predstavené v sekcií 2.3. Atribúty odtlačku sú tvorené buď priamo zistiteľnými vlastnosťami prehliadača, alebo výstupom popísaných algoritmických metód.

Vlastnosti prehliadača sú prístupné cez rozličné rozhrania, ktoré slúžia na rozdielne účely. Prístupy k týmto vlastnostiam je možné zoskupiť podľa zamerania daného rozhrania, ktoré k vlastnostiam poskytuje prístup. Takto vytvorené skupiny reprezentujú sémanticky odlišné množiny prostriedkov, ktorých využívanie má konkrétne zameranie.

Heuristiky sú postavené na predpoklade, že mnoho webových služieb má konkrétny účel. Niektoré internetové stránky slúžia na zobrazovanie multimédií, kde sa očakáva využívanie rozhraní `<video>` a `<audio>` elementov. Na druhú stranu, niektoré iné webové stránky môžu slúžiť na vzájomnú komunikáciu a využívať rozhranie *WebRTC*. Ak teda stránka pristupuje k prostriedkom nezvyčajne širokospektrálne, môže to znamenať pokus o extrakciu odtlačku prehliadača. O to viac je toto správanie podozrivé, ak stránka pristupuje k veľkému množstvu rozlične zameraných prostriedkov.

Vytvorenie skupín poskytuje možnosť abstrakcie nad konkrétnymi prostriedkami. V rámci návrhu detektujúcich heuristík je možné špecifikovať nasledovné skupiny prístupov:

- **Informácie o prehliadači** zahŕňajú základné informácie dostupné pomocou objektu `navigator`, napríklad atribúty `userAgent`, `vendor`, `product` a iné.
- **Informácie získané enumeráciou**, napríklad `MediaDevices.enumerateDevices()` alebo `navigator.plugins`.
- **Informácie o hardvéri** sú taktiež dostupné cez `navigator`, avšak obsahujú informácie o konkrétnom zariadení z pohľadu hardvérových vlastností.
- **Informácie o sieti** je možné získať pomocou objektu `navigator.connection`.
- **Informácie o obrazovke** sú informácie poskytnuté objektom `screen`.
- **Informácie o úložiskách** sa sústredia na dostupnosť objektov `localStorage`, `sessionStorage` a `indexedDB`.
- **Informácie o povoleniach** sú obsiahnuté v objekte `navigator.permissions`.
- **Informácie o časovej zóne** sú prístupné cez objekt `Date` a objekt `Intl`.
- **Informácie o podporovaných multimediálnych formátoch** je možné enumerovať objektami `HTMLAudioElement` a `HTMLVideoElement`.

- **Informácie o zvuku** sú obsiahnuté v objektoch zvukových modulov `AudioNode`.
- **Informácie o grafickom hardvéri** je možné získať cez rozhranie `WebGLRendering-Context`.
- **Informácie o presnom čase** sú dostupné v objektoch `performance` a `Date`.

Podobne ako vlastnosti prehliadača, tak aj algoritmické metódy predstavujú vlastné skupiny. Nasledovné skupiny vychádzajú z popisu algoritmických metód v podsekcií 2.3.4:

- **Enumerácia systémových písiem** zahrňuje obe metódy enumerácie cez element HTML a Canvas API.
- **Odtlačok Canvas API** zahrňuje vykresľovanie a následnú extrakciu obrázku pomocou Canvas API.
- **Odtlačok WebGL API** zahrňuje vykresľovanie a následnú extrakciu obrázku pomocou WebGL API.
- **Odtlačok WebRTC API** zahrňuje metódu získania lokálnej IP adresy.
- **Odtlačok AudioContext** zahrňuje obe spomínané metódy získania odtlačku pomocou AudioContext.

Heuristiky sa zameriavajú na kombináciu prístupov k jednotlivým spomenutým skupinám. Prístup ku skupine *vlastností prehliadača* sa považuje za vykonaný práve vtedy, keď skript pristúpil k predom špecifikovanému počtu obsiahnutých prostriedkov. Jednotlivé skupiny sa líšia v počte a type obsiahnutých prostriedkov.

Niektoré prostriedky rovnakej skupiny môžu obsahovať informácie, ktoré sa prekrývajú. Napríklad atribút `navigator.appVersion` obsahuje informácie, ktoré sú taktiež obsiahnuté v atribúte `navigator.userAgent`. Okrem toho, rôzne prostriedky môžu obsahovať rozličné množstvo identifikujúcich údajov. Tým pádom je možné pozorovať rôzne hodnoty entropie, na čo však už bolo poukázané v podsekcií 2.2.3.

Spomínané vlastnosti vytváraných skupín podmieňujú rozličný prístup ku každej z nich. Pevne stanovené kritériá pre prístup ku skupine by boli značne obmedzujúce. Kvôli tomu je pre každú skupinu a v nej obsiahnutý prostriedok nevyhnutné definovať *kritériá* podozrivého prístupu.

Kritériá prístupu presne definujú počet prístupov k prostriedku alebo skupine, ktorý je potrebný pre ich zarátanie do celkového vyhodnocovania. Vyhodnocovanie je potom založené na spočítaní prístupov k prostriedkom a skupinám, pričom prekročenie predom určených hraníc môže signalizovať pokus o získanie odtlačku prehliadača. Bližšie informácie o definícií skupín na implementačnej úrovni je možné nájsť v sekcií 4.2.

Skupina *algoritmických metód* má rovnaké vlastnosti ako ostatné skupiny. V podstate sa jedná o zoskupenia prostriedkov, ktorých kombinácia naznačuje použitie algoritmickej metódy. Taká skupina reprezentuje prístup k informáciám s vysokou hodnotou entropie. Túto skutočnosť je nutné preniesť do vytváraných heuristík.

Navrhnuté úrovne detekcie priamo odrážajú úrovne ochrany rozšírenia JSR popísaného v sekcií 3.2. Konkrétnosť heuristík stúpa spolu s úrovňou detekcie, čím je možné poskytnúť dôkladnejšiu detekciu na úkor vyššieho množstva falošne pozitívnych detekcií. Stránka je považovaná za sledujúcu odtlačkom prehliadača práve vtedy, keď sú splnené podmienky z tabuľky 3.1.

Úroveň / Skupiny	0	1	2		3	
Vlastnosti prehliadača	-	6	4	10	2	8
Algoritmické metódy	-	1	1	0	1	0

Tabuľka 3.1: Minimálne podmienky klasifikujúce stránku ako sledujúcu odtlačkom prehliadača.

Tabuľka 3.1 obsahuje minimálny počet prístupov do skupín prostriedkov, ktorý je vyžadovaný na klasifikáciu stránky ako sledujúcej. Stĺpce tabuľky reprezentujú jednotlivé úrovne, pričom 2. a 3. úroveň má dve podmienky. Pre tieto úrovne stačí, aby nastala jedna z uvedených podmienok. Napríklad, pre 2. úroveň platí, že prístup k aspoň 4 skupinám vlastností prehliadača a jednej algoritmickej metóde vedie k detekcií podozrivej činnosti. Alternatívne stačí, ak nastane aspoň 10 prístupov ku skupinám vlastností prehliadača.

Takto je možné zachytiť väčšie množstvo rozdielnych skriptov, ktoré pristupujú k vysokému množstvu nezávislých informácií. Rozdelenie do skupín umožňuje priblížiť skutočný zámer stránky, pretože sa vychádza z predpokladu, že stránka slúži na konkrétny účel a nevyžaduje rôznorodé prostriedky na fungovanie. Navyše nie je nutné inštrumentovať všetky objekty a metódy daných skupín, pretože zámer dokáže odhadnúť aj prístup k podmnožine najčastejšie využívaných prostriedkov.

Postup detekcie algoritmických metód získavania odtlačku

Detekcia priameho prístupu k vlastnostiam prehliadača je pomerne triviálna. Inštrumentácia objektov a metód, ktoré umožňujú prístup k týmto vlastnostiam, je realizovateľná pomocou *obálky*. Obálka je výsledok techniky obalenia kódu, na ktorom je postavené aj rozšírenie JSR. Obalenie navrhnuté v tejto práci nemení sémantiku obalených konštrukcií, iba zaznamenáva prístupy a potrebné metadáta.

Na druhej strane, detekcia algoritmických metód je závislá od zaznamenania série viacerých prístupov ku konkrétnym prostriedkom. Nasledovné heuristiky vychádzajú zo štúdií [10, 17], ktoré zároveň preukázali ich účinnosť.

- **Enumerácia systémových písiem (JavaScript)**
  1. Skript nastaví atribút `fontFamily` objektu `HTMLElement.style` na viac ako 20 rozdielnych hodnôt.
  2. Skript volá metódy `offsetWidth()` alebo `offsetHeight()` objektu `HTMLElement` viac ako 20-krát.
- **Enumerácia systémových písiem (Canvas API)**
  1. Skript nastaví atribút `font` objektu `CanvasRenderingContext2D` na viac ako 20 rozdielnych hodnôt.
  2. Skript volá metódu `measureText()` objektu `CanvasRenderingContext2D` viac ako 20-krát.

- **Odtlačok Canvas API**
  1. Využitie metód `fillText()` alebo `strokeText()` objektu `CanvasRenderingContext2D` k výpisu textu, kde sa nachádza aspoň 10 znakov.
  2. Aplikácia minimálne dvoch rozdielnych štýlov pomocou metód `fillStyle()` alebo `strokeStyle()` objektu `CanvasRenderingContext2D`.
  3. Extrakcia obrazových dát pomocou metód `toDataURL()`, `getImageData()` alebo `toBlob()` príslušného objektu.
- **Odtlačok WebGL API**
  1. Vytvorenie bufferu volaním metód `createBuffer()` a `bindBuffer()` v rámci objektu `WebGLRenderingContext`.
  2. Vykresľovanie volaním `vertexAttribPointer()` a `enableVertexAttribArray()` na objekt `WebGLRenderingContext`.
  3. Extrakcia obrazových dát pomocou metód `toDataURL()`, `getImageData()` alebo `toBlob()` príslušného objektu.
- **Odtlačok WebRTC API**
  1. Volanie metódy `createDataChannel()` alebo `createOffer()` v kontexte objektu `RTCPeerConnection`.
  2. Volanie metódy `onicecandidate()` alebo `localDescription()` taktiež objektu `RTCPeerConnection`.
- **Odtlačok AudioContext**
  1. Volanie minimálne jednej z nasledujúcich metód:
     - `BaseAudioContext.createOscillator()`
     - `BaseAudioContext.createDynamicsCompressor()`
     - `BaseAudioContext.destination()`
     - `OfflineAudioContext.startRendering()`
     - `OfflineAudioContext.oncomplete()`

Je nutné podotknúť, že sa nejedná o postupnosti prístupov. Jednotlivé algoritmické metódy sú definované ako skupiny prístupov, pričom nezáleží na poradí ich vykonávania. Na jednej strane by to mohlo zvýšiť presnosť detekcie, avšak na úkor vyššej réžie spojenej so zaznamenávaním časových známok a zložitejšieho vyhodnocovania. Väčšina definovaných algoritmických metód aj tak pozostáva z logicky nadväzujúcich krokov, pričom zmena poradia týchto krokov by nebola sémanticky správna.

3.3.3 Zamedzenie získavania odtlačku prehliadača

Návrh je založený na aktuálnej implementácií rozšírenia JSR a pridaní funkcionality zamedzujúcej získavaniu odtlačku prehliadača. Prvým krokom k tomuto cieľu je detekcia extrakcie odtlačku prehliadača založená na heuristikách, ktoré boli popísané v podsekcií 3.3.2. Nasleduje popis druhého kroku, ktorý je zodpovedný za zamedzenie odoslaniu extrahovaného odtlačku prehliadača.

Prehliadanie webovej stránky znamená, že sa postupne vykonávajú jednotlivé skripty, ktoré boli zo stránkou dodané. Rozšírenie ešte pred samotným spracovaním stránky obalí potrebné konštrukcie, aby bolo umožnené zaznamenávanie prístupu k monitorovaným prostriedkom. Vyhodnocovanie heuristík prebieha popri každej HTTP požiadavke, pričom výsledok tohto vyhodnotenia rozhodne či je prebiehajúcu požiadavku nutné blokovať, alebo nie. Akonáhle rozšírenie na základe heuristík rozhodne, že stránka vykonáva extrakciu odtlačku prehliadača, rozšírenie vizuálne na túto skutočnosť upozorní užívateľa.

Keďže kritériá pre pozitívne vyhodnotenie heuristík boli naplnené, všetky nasledovné HTTP požiadavky budú taktiež zablokované. Toto pravidlo platí pre všetky požiadavky, ktoré prebiehajú na pozadí bez akcie od užívateľa. Výnimku tvoria požiadavky na zmenu hlavného dokumentu, napríklad na umožnenie navigácie na novú stránku.

Blokovanie požiadaviek môže mať fatálne následky na fungovanie webových stránok. Z tohto dôvodu je nutné poskytnúť možnosť, ktorá užívateľovi umožní toto blokovanie pozastaviť.

Kapitola 4

Implementácia rozšírenia

Obsahom tejto kapitoly je popis implementácie mechanizmu, ktorý dokáže detektovať extrakciu odtlačku prehliadača a zamedzí odoslaniu tohto odtlačku na server. Sekcia 4.1 popisuje integráciu tohto mechanizmu v rámci rozšírenia JavaScript Restrictor. Sekcia 4.2 popisuje definíciu heuristík, ktoré mechanizmus využíva na posúdenie podozrivej aktivity. V sekcií 4.3 popisujem implementáciu zaznamenávania prístupov obalením príslušných prostriedkov. V poslednej sekcií 4.4 je možné nájsť konkrétny príklad implementovaného vyhodnocovania heuristík. Táto sekcia obsahuje aj popis implementácie blokovania HTTP požiadaviek po klasifikácií stránky ako sledujúcej.

4.1 Integrácia v rámci rozšírenia JavaScript Restrictor

Implementácia je realizovaná vo forme modulu, ktorý nesie názov *Fingerprinting Detection (FPD)*. Tento názov sa rovnako vyskytuje v obsiahnutej dokumentácií. JavaScript Restrictor je rozsiahlejší projekt, ktorý obsahuje viacero funkčných celkov. Niektoré funkčné celky sú nezávislé, napríklad modul *Network Boundary Shield (NBS)* [36]. Na druhú stranu, modul Fingerprinting Detection stavia na základoch rozšírenia JavaScript Restrictor. Tento modul je implementačné úzko previazaný s hlavnou logikou rozšírenia, ktorá zodpovedá za obaľovanie prostriedkov jazyka JavaScript. V rámci popisu implementácie sa bude brať ohľad iba na súčasti rozšírenia JavaScript Restrictor, ktoré priamo súvisia s implementáciou modulu Fingerprinting Detection.

Fingerprinting Detection je modul, ktorý pozostáva z viacerých logických častí. Tieto časti sú tvorené množinou súborov, ktoré boli vytvorené alebo upravené pre integráciu s rozšírením JavaScript Restrictor. Nasledujúce súbory predstavujú základnú štruktúru modulu Fingerprinting Detection.

- Súbor `manifest.json` obsahuje definíciu metadát, ktoré sú nevyhnutné pre zostavenie rozšírenia a jeho súčastí v prehliadači. Rozšírenie JavaScript Restrictor využíva osobitný súbor *manifest* vo verzií 2 pre každý z podporovaných prehliadačov. Aktuálne toto rozšírenie podporuje prehliadač Google Chrome a Mozilla Firefox. Implementácia modulu Fingerprinting Detection zaviedla nasledujúce zmeny, ktoré sú totožné pre súbory *manifest* oboch príslušných prehliadačov.

  - Pridanie skriptov `fp_code_builders.js` a `fp_detect_background.js` medzi skripty bežiace na pozadí. Poradie skriptov musí byť zachované z dôvodu logickej nadväznosti. Taktiež bol pridaný skript `fp_detect_content.js` medzi skripty s prístupom ku stránke.
  - Pridanie parametru `all_frames` s hodnotou `true` do definície skriptov s prístupom ku stránke (*content scripts*). Vďaka tomuto parametru je možne vynútiť vkladanie týchto skriptov do všetkých okien pre vyhovujúce URL adresy. To znamená, že tieto skripty budú vložené aj do všetkých obsiahnutých elementov `<iframe>`[1], ktoré predstavujú vnorený kontext prehliadača.
  - Pridanie `browsingData` do povolení rozšírenia. Toto povolenie je nutné pre manipuláciu s údajmi prehliadania, konkrétne na ich odstránenie.

- Súbor `code_builders.js` obsahuje modifikovanú implementáciu techniky obaľovania prostriedkov jazyka JavaScript, ktorá je používaná rozšírením JavaScript Restrictor. Obaľovanie je možné definovať deklaratívnym spôsobom, ktorý je bližšie popísaný na stránkach rozšírenia[2]. Výstupom tohto skriptu je kód, ktorý obaľuje deklaratívne definované prostriedky pre jednotlivé úrovne ochrany rozšírenia. Vykonané modifikácie zachovávajú pôvodný koncept obaľovania, avšak zavádzajú nové vylepšenia a opravy. Bližší popis vykonaných zmien sa nachádza v sekcií 4.3.

- Súbor `document_start.js` je skript s prístupom ku stránke, ktorý beží vždy pred samotným načítaním HTML dokumentu stránky. Úlohou tohto skriptu je vložiť obaľovací kód vygenerovaný skriptom `code_builders.js` do kontextu stránky. Výber obaľovacieho kódu závisí od úrovne ochrany, ktorá je pri danej stránke aktívna. Hoci súbor `document_start.js` neprešiel modifikáciou, považujem ho za podstatnú súčasť modulu Fingerprinting Detection.

- Súbor `fp_code_builders.js` je originálny skript modulu Fingerprinting Detection, ktorý obsahuje pomocné funkcie využívané skriptom `code_builders.js` na automatizované vytváranie obálok. Automatizované vytváranie obálok je mechanizmus, ktorý automaticky obalí prostriedky definované v heuristikách. Bližšie informácie o tomto súbore sa nachádzajú v sekcií 4.3.

- Súbor `fp_detect_background.js` je originálny skript modulu Fingerprinting Detection, ktorý je zodpovedný za detekciu získavania odtlačku prehliadača a následné blokovanie HTTP požiadaviek. Skript taktiež obsahuje funkcie, ktoré umožňujú komunikáciu s ostatnými súčasťami rozšírenia pomocou zasielania správ. Detailný popis tohto súboru je možné nájsť v sekcií 4.4.

- Súbor `fp_detect_content.js` je originálny skript modulu Fingerprinting Detection, ktorý má prístup ku stránke. Tento skript tvorí jednoduchého prostredníka medzi skriptom bežiacim na stránke (*page script*) a skriptom rozšírenia (*background script*) bežiaceho na pozadí. Existencia takého prostredníka je podmienená tým, že priama komunikácia zasielaním správ medzi skriptom stránky a skriptom rozšírenia bežiacim na pozadí nie je možná. Bližšie informácie o tomto skripte sú v sekcií 4.3.

[1] https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe
[2] https://polcak.github.io/jsrestrictor/new_wrapper.html

- Súbory `popup.js`, `popup.html` a `popup.css` definujú vzhľad a správanie vyskakovacieho okna rozšírenia, kde sú umiestnené základné ovládacie prvky. Medzi tieto prvky bol zaradený aj prepínač, ktorý umožňuje povoliť, respektíve zakázať blokovanie HTTP požiadaviek na základe detekcie získavania odtlačku prehliadača.

- Súbor `wrapping.js` je skript, ktorý poskytuje funkciu pre načítanie všetkých deklaratívne definovaných obálok do jediného globálneho objektu. Rovnaký princíp bol využitý aj pre načítanie heuristík, ktoré sú načítané do osobitného globálneho objektu hneď pri inicializácií rozšírenia.

- Súbor `fix_manifest.sh` je *bash* skript, ktorý pri zostavovaní rozšírenia príkazom `make` dynamicky modifikuje súbory *manifest*. Táto modifikácia umožňuje pridanie súborov s prefixom `wrapping`, ktoré obsahujú deklaratívne definované obálky, medzi skripty bežiace na pozadí. Rovnaký princíp je využitý pre súbory, ktoré definujú heuristiky detekcie získavania odtlačku prehliadača. Tieto súbory sú pôvodne vo formáte *JavaScript Object Notation (JSON)* [4] s koncovkou `.json`, avšak tento skript ich prevedie do súborov JavaScript s koncovkou `.js`. Počas prevodu je vložený aj kód, ktorý umožňuje pridať obsah súborov do globálneho objektu pomocou funkcie zo súboru `wrapping.js`.

- Priečinok `fp_config` zoskupuje súbory typu JSON, ktoré popisujú heuristiky použité pri vyhodnocovaní modulom Fingerprinting Detection. Nasledujúca sekcia 4.2 sa podrobne venuje popisu súborov obsiahnutých v tomto priečinku.

4.2 Deklaratívna definícia heuristík

Prvým logickým celkom modulu Fingerprinting Detection je definícia heuristík, ktoré sú využívané pri detekcií pokusu o extrakciu odtlačku prehliadača. Implementácia týchto heuristík vychádza z konceptu, ktorý bol navrhnutý v podsekcií 3.3.2. Tento koncept využíva vytváranie skupín, do ktorých sú priradené jednotlivé prostriedky podľa sémantického významu. Týmto spôsobom je možné abstrahovať jednotlivé monitorované objekty, atribúty a metódy. Prístupy je potom možné zaznamenávať na úrovni skupín, vďaka čomu je možné lepšie definovať podozrivé správanie navštívenej stránky.

Úspešnosť detekcie na úrovni pevne stanovených heuristík môže trpieť na následky pomerne nízkej flexibility. Na jednej strane môžu byť stanovené pravidlá príliš prísne, čo má za následok vysoký počet *falošne negatívnych* detekcií. Na druhej strane môžu byť pravidlá nadmieru benevolentné s vysokým počtom *falošne pozitívnych* detekcií. Nájdenie správnej hranice pri nastavovaní pravidiel je kľúčové pri tomto spôsobe detekcie. Okrem toho, prostredie internetu sa stále vyvíja, a to vysokým tempom. Je možné pozorovať pravidelné zmeny v podpore nových rozhraní jednotlivými prehliadačmi, kde niektoré rozhrania sú úplne nové a niektoré len nahrádzajú tie zastaralé. Nové rozhrania môžu priniesť nové spôsoby na získavanie odtlačku prehliadača. To však znamená, že pevne stanovené heuristiky by postupom času stratili významnú časť pokrytia monitorovaných prostriedkov.

Tieto problémy priamo rieši *deklaratívny prístup* k definícií heuristík. Tento mechanizmus využíva súbory formátu JSON, ktoré obsahujú informácie a skupinách a priradených prostriedkoch. Vývojár, ale aj pokročilý užívateľ, je schopný definovať vlastné pravidlá detekcie. V prípade nových prostriedkov, ktoré môžu byť použité v rámci odtlačku prehliadača, je možné tieto prostriedky jednoducho pridať. Deklaratívny prístup poskytuje nástroj pre jednoduchú aktualizáciu a dôkladné nastavenie detekčných heuristík.

Definícia heuristík je rozdelená na dva typy súborov, ktoré sa nachádzajú v priečinku `fp_config`:

- Súbory `wrappers-lvl_X.json` obsahujú definíciu obaľovaných prostriedkov a ich priradenie do skupín.
- Súbory `groups-lvl_X.json` obsahujú definíciu skupín a kritérií, ktoré sú potrebné pre započítanie prístupu ku skupine.

Súbory sú definované pre jednotlivé úrovne rozšírenia JavaScript Restrictor. Úrovne sú špecifikované znakom `'X'` priamo v názve súboru za podreťazcom `"lvl"`. Jeden súbor môže pokrývať definíciu pre viacero úrovní zároveň, a to jednoduchým pridaním dodatočného označenia úrovne do názvu súboru. Definíciu úrovní v názve súboru je nutné patrične oddeliť znakom `'_'`. Napríklad súbor s názvom `wrappers-lvl_1_2.json` bude obsahovať definíciu obaľovaných prostriedkov pre úrovne 1 a 2. V nasledujúcich podsekciách je detailne popísaná štruktúra týchto súborov.

4.2.1 Definícia obálok

Definícia *obálok* (`wrappers`) sa nachádza v súbore `wrappers-lvl_X.json`. Tento súbor na najvyššej úrovni reprezentuje *pole objektov*. Každý z týchto objektov predstavuje jeden obaľovaný *prostriedok* (`resource`). Príklad štruktúry tohto súboru je znázornený v kóde 4.1.

```
[
    {
        "resource": "navigator.userAgent",
        "type": "property",
        "groups": []
    },
    {
        "resource": "CanvasRenderingContext2D.prototype.fillStyle",
        "type": "function",
        "groups": []
    }
]
```

Kód 4.1: Príklad súboru `wrappers-lvl_X.json`, ktorý obsahuje definíciu obaľovaných prostriedkov.

Každý objekt obaľovaného prostriedku musí obsahovať 3 povinné atribúty:

- Atribút `"resource"` obsahuje celý názov obaľovaného prostriedku spolu s rodičovskými objektami. Objekt `window` je možné v tomto kontexte vynechať.
- Atribút `"type"` obsahuje reťazec, ktorý vyjadruje typ obaľovaného prostriedku.
  - Hodnota `"property"` označuje *atribút objektu*.
  - Hodnota `"function"` označuje *metódu objektu*.

- Atribút `"groups"` definuje príslušnosť obaľovaného prostriedku do skupín. Jedná sa o *pole objektov*, pričom každý objekt bližšie špecifikuje postavenie prostriedku v rámci priradenej skupiny.

Definícia príslušnosti prostriedku v skupine je založená na objektoch, ktorých štruktúra závisí od typu obaľovaného prostriedku. Príklad objektu, ktorý priraďuje obalený prostriedok typu `"property"` do skupiny, je naznačený kódom 4.2. Jeden prostriedok môže prináležať viacerým skupinám, avšak v každej skupine môže mať rozdielne postavenie. Za týmto účelom je možné pridať do atribútu `"groups"` ľubovoľné množstvo objektov.

```
{
    "group": "BrowserInfo",
    "property": "set",
    "arguments": "diff",
    "criteria": []
}
```

Kód 4.2: Príklad objektu, ktorý priraďuje obaľovaný prostriedok do skupiny `BrowserInfo`. Ďalej špecifikuje dodatočné informácie, ako obalenie typu *setter* a zaznamenávanie počtu prístupov s rozdielnymi argumentami.

Objekty definujúce príslušnosť do skupiny obsahujú nasledovné atribúty:

- Atribút `"group"` je **povinný** atribút, ktorý obsahuje meno pridelenej skupiny. Názov skupiny vyplýva z jej definície v súbore `groups-lvl_X.json`.
- Atribút `"property"` je **voliteľný** atribút, ktorý špecifikuje formu obalenia pre prostriedky typu `"property"`.
  - Hodnota `"get"` označuje *getter* atribútu, kedy sa obaľuje prístup k hodnote atribútu.
  - Hodnota `"set"` označuje *setter* atribútu, kedy sa obaľuje nastavenie hodnoty atribútu.

  Nastavenie atribútu `"property"` má význam iba v prípade, že obaľovaný prostriedok je typu `"property"`. V opačnom prípade je hodnota tohto atribútu ignorovaná. V prípade vynechania je nastavená implicitná hodnota atribútu `"property"` na hodnotu `"get"`.
- Atribút `"arguments"` je **voliteľný** atribút, ktorý definuje zaznamenávanie prístupu podľa vstupných argumentov. Tento atribút má význam v kontexte prostriedkov, ktoré obsahujú vstupné parametre. Pokiaľ tento atribút nie je špecifikovaný, zaznamenávajú sa všetky prístupy bez ohľadu na vstupné argumenty. Ak je atribút `"arguments"` špecifikovaný, prístup sa zaznamenáva iba pre vyhovujúce argumenty. Tento atribút môže obsahovať nasledujúce hodnoty:
  - Hodnota `"diff"` je reťazec, ktorým je možné nastaviť zaznamenávanie počtu rozdielnych vstupných argumentov. To znamená, že počet prístupov k prostriedku bude závislý na počte rôznych argumentov, s ktorými bol realizovaný prístup k obalenému prostriedku.

  - Hodnota `"same"` je reťazec, ktorým je možné nastaviť zaznamenávanie maximálneho počtu prístupov s rovnakými argumentami. Týmto spôsobom je možné zaznamenať napríklad to, že funkcia bola viacnásobne volaná s rovnakým vstupom.
  - *Regulárny výraz*, ktorým je možné zadefinovať konkrétne hodnoty vstupných argumentov. Regulárny výraz je založený na štandardnej implementácií[3] v jazyku JavaScript. Definícia regulárneho výrazu má formu pola, ktoré predstavuje vstupné argumenty konštruktoru `RegExp()`. Pre príklad je možné uviesť výraz `[".{10,}"]`, ktorým je možné obmedziť zaznamenávanie iba na argumenty s počtom znakom väčším ako 10.

- Atribút `"criteria"` je **voliteľný** atribút, ktorým je možné priradiť obaľovanému prostriedku **váhu** (*weight*). Táto váha je platná iba v kontexte priradenej skupiny. Hodnota atribútu `"criteria"` je tvorená poľom, ktoré obsahuje jednotlivé objekty kritérií. Príklad objektu kritéria je znázornený kódom 4.3. Každý prostriedok môže mať definovaný ľubovoľný počet kritérií. Ak je atribút `"criteria"` vynechaný, aplikujú sa implicitné kritéria definované v kóde 4.4. Váhy, ktoré sú na základe kritérií priradené prostriedkom, sa využívajú v procese vyhodnocovania heuristík. Tento proces a jeho implementácia je bližšie popísaná v sekcií 4.4.

```
{
    "value":20,
    "weight":1
}
```

Kód 4.3: Príklad objektu kritéria, ktorý danému prostriedku priradí váhu 1 práve vtedy, keď je k prostriedku realizovaných aspoň 20 prístupov.

```
{
    "value":1,
    "weight":1
}
```

Kód 4.4: Implicitný objekt kritéria, ktorý danému prostriedku priradí váhu 1 práve vtedy, keď sa k prostriedku pristúpilo aspoň 1-krát.

4.2.2 Definícia skupín

Definícia *skupín* (`groups`) sa nachádza v súbore `groups-lvl_X.json`. Tento súbor obsahuje rekurzívnu definíciu skupín, kde každá skupina je reprezentovaná vlastným objektom. Na najvyššej úrovni sa nachádza jediný objekt, ktorý predstavuje definíciu *koreňovej skupiny* (*root group*). Každá skupina môže obsahovať ľubovoľný počet podskupín alebo priamo obaľovaných prostriedkov. Týmto spôsobom je možné definovať stromovú štruktúru pozostávajúcu zo skupín a prostriedkov. Pomocou tejto štruktúry je možné vyjadriť veľké množstvo pravidiel, ktoré sa uplatňujú pri vyhodnocovaní procesu detekcie. Príklad definície koreňovej skupiny je znázornený kódom 4.5.

[3] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

```
{
    "name":"FingerprintingActivity",
    "description": "Root group of fingerprinting detection heuristics",
    "criteria":[
    {
        "value": 2,
        "weight": 1
    },
    "groups": []
}
```

Kód 4.5: Príklad definície koreňovej skupiny `FingerprintingActivity`. Váha tejto skupiny rozhoduje o celkovom výsledku vyhodnocovania heuristík a detekcie.

Každý objekt skupiny pozostáva z nasledovných atribútov:

- Atribút `"name"` je **povinný** atribút, ktorý obsahuje názov definovanej skupiny. Názov musí byť jedinečný v rámci všetkých definovaných skupín.
- Atribút `"description"` je **voliteľný** atribút, ktorý obsahuje popis definovanej skupiny.
- Atribút `"criteria"` je **voliteľný** atribút, ktorým je možné skupine priradiť **váhu** (*weight*). Hodnota atribútu `"criteria"` je tvorená poľom kritérií, ktoré majú rovnakú štruktúru ako kritériá obaľovaných prostriedkov v kóde 4.3. Každá skupina môže mať definovaný ľubovoľný počet kritérií. Vynechanie atribútu `criteria` znamená použitie implicitných kritérií z kódu 4.4. Skupiny tvoria jadro rozhodovacej logiky, ktorá slúži na vykonávanú detekciu. Pre zvýšenie vyjadrovacej sily takto definovaných heuristík, skupiny naviac podporujú nasledovné alternatívne definície kritérií:
  - Kritériá `"value"`, ktoré priraďujú skupine váhu na základe aktuálnej hodnoty váh všetkých *priamych potomkov* (podskupín a obsiahnutých prostriedkov).
  - Kritériá `"percentage"`, ktoré priraďujú skupine váhu na základe aktuálnej percentuálnej hodnoty váh všetkých priamych potomkov. Percentuálny podiel sa berie zo súčtu maximálne dosiahnuteľných váh týchto potomkov.
  - Kritériá `"access"`, ktoré priraďujú skupine váhu na základe počtu prístupov k priamym potomkom. Narozdiel od predchádzajúcich kritérií, tieto kritéria berú v úvahu absolútny počet prístupov k obsiahnutým prostriedkom namiesto ich aktuálnej váhy. Týmto spôsobom je možné vyjadriť skupiny prostriedkov, pri ktorých je podstatný celkový počet prístupov k ľubovolnému prostriedku skupiny.
- Atribút `"groups"` je **voliteľný** atribút, ktorý pre danú skupinu umožňuje definíciu ľubovoľného počtu *podskupín*. Každá podskupina je tvorená štrukturálne rovnakým objektom skupiny.

4.3 Zaznamenávanie obalených prostriedkov

JavaScript Restrictor využíva techniku obaľovania prostriedkov za účelom modifikácie ich tradičného správania. Pôvodne obaľované prostriedky sú explicitne definované v súboroch

`wrappingS-XYZ.js`, kde `"XYZ"` predstavuje pomenovanie obaľovaných API podľa štandardu[4] ich predstavenia. Z deklaratívnej definície obaľovaných prostriedkov vznikne kód, ktorý tieto prostriedky patrične obalí. Kód je následne vkladaný na navštívené stránky.

Tento mechanizmus využíva aj modul Fingerprinting Detection. Obaľované prostriedky sú dostupné v súboroch `fp_config/wrappers-lvl_X.js` pre jednotlivé úrovne ochrany. Zostavenie rozšírenia príkazom `make` zavolá skript `fix_manifest.sh`, ktorý tieto súbory transformuje do súborov `common/wrappingX-XYZ.js`. Podreťazec `"XYZ"` je v tomto prípade nahradený názvom pôvodného súboru. Skript následne vytvorené súbory pridá do súborov *manifest* medzi `background scripts`. Novo vytvorené súbory sú taktiež doplnené o kód, ktorý ich obsah uloží vo forme reťazca do globálneho objektu `fp_config_code`. Týmto spôsobom je možné nahrať aktuálnu konfiguráciu modulu ešte pred generovaním obaľovacieho kódu.

4.3.1 Generovanie obaľovacieho kódu

Pri inicializácií rozšírenia beží skript `fp_code_builders.js`, ktorý načítanú konfiguráciu modulu spracuje do objektu `fp_levels`. Tento objekt obsahuje všetky konfiguračné súbory prevedené do objektov JavaScript a zoradené podľa príslušných úrovní.

Generovanie obaľovacieho kódu je vzápätí zahájené skriptom `code_builders.js`. Skript najprv obalí explicitne definované prostriedky zo súborov `wrappingS-XYZ.js`. Následne je nutné obaliť zvyšné prostriedky, ktoré neboli explicitne definované. Na tento účel sa využíva metóda `fp_wrappers_create()`, ktorá má na vstupe pole explicitne definovaných prostriedkov. Hlavnou úlohou tejto metódy je generovať štandardnú deklaratívnu definíciu[5] obálok pre všetky prostriedky, ktoré doposiaľ neboli obalené. Definícia tejto funkcie a jej pomocných funkcií je v súbore `fp_code_builders.js`. Výstupom funkcie `fp_wrappers_create()` je naplnenie a vrátenie objektu `new_build_wrapping_code`, ktorý má rovnakú štruktúru ako objekt `build_wrapping_code` obsahujúci definíciu explicitne definovaných obálok. Pre všetky prostriedky výstupného objektu je generovaný obaľovací kód pomocou funkcie `build_code()`.

Obaľovanie modulom Fingerprinting Detection má za cieľ obaliť špecifikované prostriedky, avšak zachovať ich pôvodnú funkciu. Týmto spôsobom je umožnené pridať kód, ktorý bude zaznamenávať prístupy k týmto prostriedkom, a to bez funkčných obmedzení. Aby bolo možné definované prostriedky obaľovať jednotným spôsobom, bolo nutné aplikovať sériu úprav pôvodného generátora kódu. Spôsob obaľovanie funkcií sa oproti pôvodnej verzií výrazne nezmenil. V princípe sa jedná o doplnenie deklaratívnej definície obálok o kód, ktorý umožňuje zaznamenávanie prístupov.

Výraznejšie zmeny sú badateľné pri spôsobe obaľovania atribútov objektov. Atribúty môžu byť definované dvoma spôsobmi [26] na základe ich deskriptora:

- Atribúty definované *dátovým (data) deskriptorom.*
- Atribúty definované *prístupovým (accessor) deskriptorom.*

Atribúty definované prístupovým deskriptorom obsahujú vo svojom deskriptore funkcie `get` a `set`, ktoré slúžia na získanie hodnoty atribútu, respektíve na jeho nastavenie. Obalenie týchto funkcií umožňuje zaznamenávať prístupy k takto definovaným atribútom. Explicitná definícia umožňuje pri obaľovaní atribútov nastaviť *getter* aj *setter* na hodnotu alebo

[4] https://github.com/pes10k/web-api-manager/tree/master/sources/standards
[5] https://polcak.github.io/jsrestrictor/new_wrapper.html

funkciu. Táto hodnota je nastaviteľná v atribúte `property_value` deklaratívnej definície obálok. Pri vrátení originálnej hodnoty alebo hodnoty špecifikovanej explicitným obaľovaním je nutné overiť či sa jedná o funkciu, alebo priamo hodnotu. Toto overenie je znázornené kódom 4.6.

```
if (typeof (property_value) === 'function') {
    return (property_value).bind(this)(...args);
}
else {
    return property_value;
}
```

Kód 4.6: Overenie typu hodnoty `property_value` a voľba následnej akcie. Ak sa jedná o funkciu, volaj danú funkciu v pôvodnom kontexte s pôvodnými argumentami. Ak sa jedná o hodnotu iného dátového typu, vráť túto hodnotu.

Medzi ďalšie zmeny je možné zaradiť nový režim, ktorý je implementovaný v rámci funkcie `generate_object_properties()` generujúcej kód obaľujúci atribúty. Režim je možné aktivovať nastavením parametru `only_fp_mode` na hodnotu `true`. Tento režim rieši atribút `apply_if`, ktorý je možné použiť pri explicitnej definícií na podmienené obaľovanie. Využitie tohto atribútu by však znamenalo, že prostriedok bude obalený, len ak určitá podmienka bude splnená. To by znamenalo, že by daný prostriedok nemusel byť obalený ani kódom, ktorý zaznamenáva prístupy. Režimom `only_fp_mode` je vygenerovaný kód, ktorý vždy obaľuje na účely zaznamenávania prístupu.

Pridaná bola aj možnosť *vynúteného obalenia* prostriedku, ak tento prostriedok nie je definovaný. Túto možnosť je nutné definovať priamo v explicitnej definícií obaľovania alebo pri definícií prostriedkov v konfiguračných súboroch modulu Fingerprinting Detetction. Definícia je realizovaná pridaním atribútu `force_wrapping` s hodnotou `true` do definície obaľovania prostriedku. Jedná sa o pokročilú funkciu, ktorá musí byť využívaná opatrne s vedomím, že môže obmedziť niektoré funkcie stránky alebo prehliadača. Aktuálne je táto funkcia využitá pri obaľovaní atribútu `fontFamily` objektu `CSSStyleDeclaration`. Keďže tento objekt vzniká nanovo s každou inštanciou objektu `HTMLElement`, obalenie atribútov je možné cez jeho prototyp. Atribút `fontFamily` však nie je definovaný na prototype tohto objektu a vzniká taktiež dynamicky. Aby bolo umožnené zaznamenávanie prístupov, jedným z riešení je definícia tohto atribútu do prototypu objektu. Týmto spôsobom síce zamedzíme pôvodnú funkcionalitu tohto atribútu, avšak sme schopný zaznamenávať prístupy k nemu zo všetkých inštancií objektu `CSSStyleDeclaration`. Vytváranie atribútu je realizované predom vytvoreným deskriptorom, ktorý je vyjadrený kódom 4.7

```
descriptor = { // Descriptor for newly created property
    get: () => { return this.property_name },
    set: (val) => { this.property_name = val },
    configurable: false,
    enumerable: true
};
```

Kód 4.7: Deskriptor, ktorý je použitý pri vytváraní nového atribútu pri aktivovaní možnosti `force_wrapping`.

Na konci každého obalenia je nutné objekt *zamraziť* metódou `Object.freeze()`[6]. Týmto spôsobom je možné zamedziť jeho následnej úprave. Prístupy k atribútom niektorých objektov je možné zaznamenávať priamo obalením funkcií `get` a `set` na ich prototype. Toto je možné použiť napríklad v prípade, že inštancia objektu na danom atribúte nemá definovaný deskriptor. Objekt prototypu nie je možné priamo zamraziť, a preto je nutné nastaviť atribút na prototype na `configurable:false` a zároveň zamraziť inštanciu tohto objektu, ak existuje pod rovnakým menom na objekte `window`.

4.3.2 Zaznamenávanie prístupov

Zaznamenávanie obalených prostriedkov je realizované pomocou mechanizmu *správ*. Obaľovanie prostriedkov skriptom `code_builder.js` automaticky vkladá potrebné konštrukcie, ktoré sú zodpovedné za vytvorenie a odoslanie správy o prístupe. Správy sú odosielané príslušnému oknu `window` pomocou funkcie `postMessage()`. Vkladaná konštrukcia, ktorá vytvára a odosiela záznamy o prístupoch, je znázornená kódom 4.8.

Správa o prístupe obsahuje položku `purpose`, na základe ktorej je možné identifikovať typ správy. Položka `enabled` je definovaná premennou `fp_enabled`. Táto premenná slúži na povolenie zaznamenávania prístupu až po vykonaní celého obaľovacieho kódu. V opačnom prípade sa môžu započítať aj prístupy, ktoré sú vykonané obaľovacím kódom. Položka `resource` obsahuje názov prostriedku a položka `type` obsahuje jeho typ. Nakoniec položka `args` obsahuje pole vstupných argumentov, ktoré sú prevedené na reťazce.

```
window.top.postMessage({
    purpose: "fp-detection",
    enabled: fp_enabled,
    resource: "navigator.userAgent",
    type: "get",
    args: Array.from(arguments).map(x => JSON.stringify(x))
}, "*");
```

Kód 4.8: Kód, ktorý je vkladaný v rámci obaľovania funkcií a atribútov. Týmto spôsobom kód komunikuje s vloženým *content* skriptom `fp_detect_content.js`.

Takto vytvorené správy sú odoslané hlavnému objektu `window`, teda hlavnému oknu v danej karte. Správy sú následne prijímané skriptom v súbore `fp_detect_content.js`. Tento súbor je vložený do všetkých elementov, ktoré predstavujú osobitný kontext, ako napríklad elementy `<iframe>`. Správu o prístupe obdrží iba ten skript, ktorý je vložený v hlavnom okne. Súbor `fp_detect_content.js` obsahuje *event listener*, ktorý čaká na správy typu `"fp-detection"`. Všetky také správy sú následne preposlané *background* skriptu `fp_detect_background.js` na ďalšie spracovanie. Táto implementácia je nutná z toho dôvodu, že skripty stránky nemajú nástroje na priamu komunikáciu s *background* skriptom.

Pri testovaní bolo zistené, že veľmi veľký počet za sebou nasledujúcich správ má vysokú réžiu. Réžia spojená zo zasielaním správ má tendenciu rásť exponenciálne. Tento problém bol vyriešený zavedením počítadla `fp_counter_call`, ktoré je definované v lokálnom rozsahu obalených prostriedkov. Aktuálne je počet správ na obalený prostriedok obmedzený na 1000. Predpokladá sa, že heuristiky nebudú vyžadovať zaznamenávanie počtu prístupov pre vyššie hodnoty.

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

Skript `fp_detect_background.js` obsahuje *event listener*, ktorý čaká na správy typu `"fp-detection"` odoslané *content* skriptom. Skript po prijatí správy túto správu spracuje a relevantné informácie uloží do *databázy prístupov*. Databáza prístupov je reprezentovaná globálnym objektom `fpDb`, ktorý je taktiež definovaný v tomto skripte. Štruktúra objektu `fpDb` je zobrazená v kóde 4.9.

```
resource_name: {
    tab_id: {
        resource_type: {
            arguments: [
                "star": 2,
                "wars": 1
            ],
            total: 3
        }
    }
}
```

Kód 4.9: Štruktúra objektu `fpDb`, ktorý obsahuje databázu prístupov.

Pre každý prostriedok je zaznamenané to, v ktorej karte prehliadača nastal. Karta je určená jedinečným identifikátorom, ktorý jej priradí sám prehliadač. Pre každý typ prostriedku sú definované záznamy celkového počtu prístupov a pole argumentov, ktoré obsahuje počet prístupov pre jednotlivé vstupné argumenty. Tieto záznamy sú následne využívané pri vyhodnocovaní heuristík a detekcií extrakcie odtlačku prehliadača.

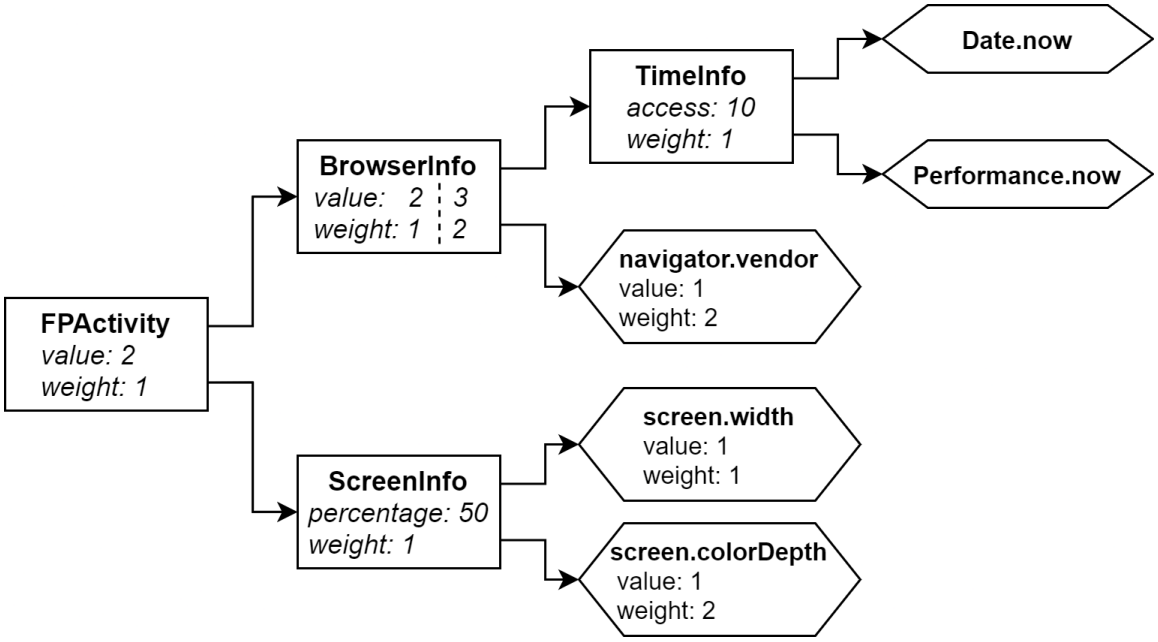
4.4 Vyhodnocovanie pomocou heuristík

Vyhodnocovanie heuristík sa vykonáva pri každej vykonanej HTTP požiadavke. Súbor `fp_detect_background.js` obsahuje funkciu `evaluateGroups()`, ktorá toto vyhodnocovanie inicializuje. Táto funkcia má parameter `tabId`, ktorý predstavuje identifikátor karty prehliadača. Vyhodnocovanie prebieha v kontexte kariet, čím je možné pokryť všetky požiadavky načítanej stránky. To znamená, že nezáleží na aktuálnej adrese okna ani na adrese dodatočne načítaných prvkov daného okna. Týmto spôsobom je možné zahrnúť všetky obsiahnuté tretie strany do vyhodnocovania.

Vyhodnocovanie využíva zaznamenané informácie o prístupoch dostupné v objekte `fpDb`. Záznamy v tomto objekte sú aktualizované vždy pri zmene stavu zaznamenávanej karty. Pri udalostiach `tabs.onRemoved` a `tabs.onUpdated` je obsah záznamov pre danú kartu vyčistený. V prípade udalosti `tabs.onUpdated` je čistenie vykonané vždy pred načítaním novej stránky. Vďaka tomu je možné nezávisle zaznamenávať prístupy pre jednotlivé načítania stránky.

4.4.1 Príklad procesu vyhodnocovania definovaných heuristík

Implementáciu vyhodnocovacej logiky je pre lepšie pochopenie vhodné demonštrovať na konkrétnom príklade heuristík. Heuristiky tohto príkladu je možné znázorniť vo forme grafu na obrázku 4.1. Tento graf predstavuje stromovú štruktúru, ktorá vzniká rekurzívnym definovaním skupín v konfiguračných súboroch. Detailný popis týchto súborov sa nachádza v sekcií 4.2.



Obr. 4.1: Príklad heuristík, ktoré je možné definovať konfiguračnými súbormi modulu Fingerprinting Detection. Obdĺžniky predstavujú príklady skupín zo súborov `groups-lvl_X.json`. Šesťuholníky predstavujú príklady obaľovaných prostriedkov zo súborov `wrappers-lvl_X.json`.

Hlavnou metrikou vyhodnocovania heuristík sú *kritéria* skupín a prostriedkov. Kritéria definujú hodnotu *váhy*, ktorá im bude pridelená pri vyhodnocovaní. Váhy sú prideľované na základe podmienok stanovených v objektoch kritérií. Podmienky závisia od váh a prístupov priamych potomkov skupiny. Váhy prostriedkov sú prideľované iba na základe ich prístupov, pretože prostriedky nemajú potomkov. V grafe na obrázku 4.1 sú kritériá zobrazené pod názvom skupiny či prostriedku.

Vyhodnocovanie začína koreňovou skupinou `FPActivity`. Na túto skupinu sa volá funkcia `evaluateGroupsCriteria()`, ktorá prideľuje váhu skupine na základe jej priamych potomkov. Keďže potomkovia ešte nemajú pridelenú váhu, rovnaká funkcia sa volá na všetkých potomkov typu skupina. Ak sa jedná o potomka typu prostriedok, potom sa volá funkcia `evaluateResourcesCriteria()`, ktorá prostriedku priradí váhu v kontexte rodičovskej skupiny. Tieto volania sa vykonávajú rekurzívne a hodnoty váh sú priraďované pri spätnom návrate. Hodnota takto získanej váhy koreňovej skupiny `FPActivity` signalizuje podozrivé správanie podľa definovaných heuristík.

Lepšie pochopenie fungovania heuristík je možné pozorovať pri návrate rekurzívnych volaní. Váhy jednotlivých skupín sú prideľované nasledovne:

- Skupina `TimeInfo` obsahuje prostriedky prístupu k presnému času. Kritériá tejto skupiny obsahujú kľúčové slovo `"access"`, čím sa dáva najavo, že váha skupiny je závislá na prístupoch a nie váhach priamych potomkov. V aktuálnej konfigurácií to znamená, že skupina `TimeInfo` nadobudne váhu 1, ak bude realizovaných aspoň 10 prístupov k týmto prostriedkom. Zároveň však nezáleží na konkrétnom prostriedku, pretože sa berie súčet prístupov zo všetkých obsiahnutých prostriedkov. Napríklad, ak počet prí-

stupov `Date.now` je 5 a počet prístupov `Performance.now` je 7, potom je výsledný počet prístupov 12, čo spĺňa kritérium skupiny pre priradenie váhy.

- Skupina `BrowserInfo` obsahuje prostriedok `navigator.vendor` a skupinu `TimeInfo`. Táto skupina zároveň obsahuje dva kritéria, ktoré pomocou kľúčového slova `"value"` určujú, že priradenie váhy je závislé od súčtu aktuálnych váh priamych potomkov. Skupina `BrowserInfo` nadobudne váhu 1, ak súčet váh priamych potomkov bude 2. Toto je možné dosiahnuť jediným prístupom k prostriedku `navigator.vendor`. V prípade, že skupina `TimeInfo` nadobudne váhu 1, výsledná váha skupiny `BrowserInfo` bude 2.

- Skupina `ScreenInfo` obsahuje prostriedky, ktoré poskytujú prístup k informáciám o obrazovke. Táto skupina obsahuje kritérium `"percentage"`, ktoré priraďuje váhu na základe percentuálneho podielu nadobudnutých váh priamych potomkov. Percentuálny podiel je vypočítaný zo súčtu maximálne dosiahnuteľných váh všetkých priamych potomkov. Hodnota 50 definuje kritérium, kde súčet nadobudnutých váh priamych potomkov musí byť aspoň 50 % z dosiahnuteľného maxima pre pridelenie váhy 1. V tomto konkrétnom prípade je maximum váh priamych potomkov 3, pričom 50 % tvorí hodnotu 1,5. Táto hodnota sa zaokrúhľuje na celé číslo 2. Prístup k prostriedku `screen.width` má váhu 1, čo nestačí na splnenie kritéria. Prístup k prostriedku `screen.colorDepth` má váhu 2, čím spĺňa toto kritérium a skupina `ScreenInfo` môže nadobudnúť váhu 1.

- Skupina `FPActivity` je koreňová skupina, ktorej váha vyjadruje podozrenie na získavanie odtlačku prehliadača. Táto skupina má kritérium `"value"`, ktoré závisí na súčte váh priamych potomkov. Skupina `FPActivity` môže nadobudnúť váhu 1 v nasledovných prípadoch:
  - Podskupiny `BrowserInfo` a `ScreenInfo` nadobudli váhu 1.
  - Podskupina `BrowserInfo` nadobudla váhu 2.

4.4.2 Zamedzenie odoslania získaného odtlačku

Skript `fp_detect_background.js` obsahuje *event listener*, ktorý odchytáva všetky vykonávané HTTP požiadavky. Funkcia `cancelCallback()` je volaná pre každú odchytenú požiadavku a na základe výsledku vyhodnocovania heuristík rozhoduje o jej blokovaní. Užívateľ je o blokovaní informovaný pomocou notifikácie.

Požiadavky typu `"main_frame"` vyžadujú vyčistenie databázy prístupov `fpDb` ešte pred vyhodnocovaním heuristík. Prehliadač Chrome totižto najprv vygeneruje udalosť požiadavky `webRequest.onBeforeRequest` a až potom udalosť `tabs.onUpdated` aktualizácie karty. Kvôli tomu sa databáza prístupov nestihne včas vyčistiť, čo má za následok zablokovanie nasledujúcej užívateľskej činnosti.

Pri blokovaní stránky prebieha aj vymazávanie dát, ktoré boli v prehliadači touto stránkou uložené. To zamedzuje zneužitiu lokálnej pamäte prehliadača na uchovanie získaného odtlačku za účelom dodatočného odoslania. Pri testovaní bolo zistené, že niektoré stránky naozaj ukladajú odtlačok aj v lokálnej pamäti. Pokiaľ by tieto dáta neboli vymazané, následné navštívenie sledujúcej stránky by nebolo zablokované, pretože stránka už nepotrebuje pristupovať k prostriedkom na získanie odtlačku prehliadača. Tento odtlačok si načíta

priamo z pamäte prehliadača a voľne odošle na server. Vymazávanie dát prehliadania je vykonávané na úrovni domén pomocou rozhrania `browsingData`[7]. Metóda `remove()` má na vstupe objekt, ktorý reprezentuje vymazávané typy dát. Objekt použitý pri implementácií v prehliadači Chrome je znázornený kódom 4.10.

```
{
    "cacheStorage": true,
    "cookies": true,
    "fileSystems": true,
    "indexedDB": true,
    "localStorage": true,
    "serviceWorkers": true,
    "webSQL": true
}
```

Kód 4.10: Objekt `DataTypeSet`, ktorým je možné definovať vymazávané typy dát prehliadača.

Okrem týchto dát je vhodné vymazať aj dáta v `sessionStorage` a `localStorage` všetkých okien v danej karte. Toto je realizované poslaním správy skriptu `fp_detect_content`, ktorý na túto správu reaguje vymazaním potrebných dát.

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

Kapitola 5

Testovanie

Táto kapitola zahrňuje testovanie implementácie modulu *Fingerprinting Detection* a definovaných heuristík. Sekcia 5.1 sa venuje demonštrácií funkčnosti na nástrojoch získavajúcich odtlačok prehliadača. Táto sekcia obsahuje názorné ukážky ako modul reálne funguje. V nasledujúcej sekcií 5.2 sa nachádzajú testy na obyčajných stránkach, kde sa vyhodnocuje úspešnosť detekcie v reálnom prostredí. V záverečnej sekcií 5.3 diskutujem o obmedzeniach implementácie a navrhujem možné zlepšenia.

5.1 Testovanie funkčnosti implementácie

Táto sekcia sa venuje popisu funkčnosti modulu Fingeprinting Detection a demonštruje jeho možnosti. Detekcia získavania odtlačku prehliadača úzko súvisí s nastavením vyhodnocovacích heuristík. Z tohto dôvodu je vhodné najprv poukázať na funkčnosť tohto modulu v rámci vhodne zvolených testovacích stránok. Za týmto účelom boli využité verejne dostupné nástroje *Am I Unique*[1] a *Cover Your Tracks*[2]. Tieto nástroje boli vytvorené počas viacerých štúdií skúmajúcich odtlačky prehliadača. Oba nástroje využívajú odlišné skripty na extrakciu odtlačku prehliadača.

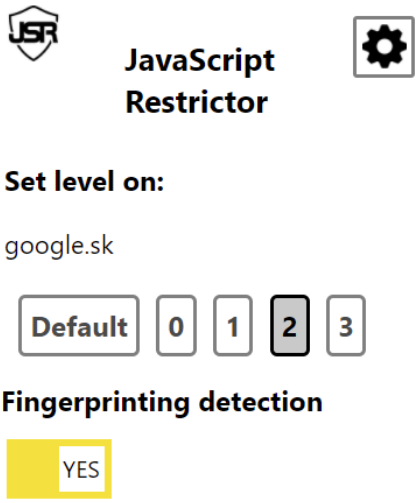
5.1.1 Demonštrácia funkčnosti

Demonštrácia funkčnosti je založená na porovnaní správania testovacích stránok pri vypnutí a zapnutí modulu Fingerprinting Detection. Prístup k týmto stránkam je v oboch prípadoch realizovaný s aktívnym rozšírením JavaScript Restrictor.

Testovaný modul obsahuje možnosť osobitnej deaktivácie vo vyskakovacom okne rozšírenia. Po kliknutí na ikonu rozšírenia JavaScript Restrictor je možné modul aktivovať alebo deaktivovať. Ukážka vyskakovacieho okna a prepínača modulu je znázornená na obrázku 5.1.

Výhodou tohto prepínača je to, že zmeny sú aplikované okamžite. Nie je nutné obnoviť kartu prehliadača a ani zobrazovaný dokument. Toto umožňuje dočasné vypnutie tejto funkcie v prípade, že funkčnosť navštívenej stránky je negatívne ovplyvnená blokovaním HTTP požiadaviek.

[1] https://amiunique.org/fp
[2] https://coveryourtracks.eff.org/



Obr. 5.1: Vyskakovacie okno rozšírenia JavaScript Restrictor, ktoré obsahuje prepínač modulu Fingerprinting Detection.

Prístup na testovaciu stránku bez modulu Fingerprinitng Detection

Prístup na stránku `amiunique.org` je realizovaný pomocou prehliadača Google Chrome s aktívnym rozšírením JavaScript Restrictor. Modul Fingerprinting Detection je vypnutý prepínačom vyskakovacieho okna. Po navigácií na adresu `amiunique.org/fp` sa začne vykonávať extrakcia odtlačku prehliadača. Výsledná stránka zahrňujúca získaný odtlačok je zobrazená na obrázku 5.2.

Javascript attributes

Attribute	Similarity ratio All time	Value
User agent	0.02%	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
Platform	39.25%	Win32
Cookies enabled	75.69%	yes
Timezone	13.61%	-120
Content language	0.02%	sk-SK,sk,cs,en-US,en

Obr. 5.2: Prístup na stránku `amiunique.org/fp` s vypnutým modulom Fingerprinting Detection.

Ako je možné vidieť na obrázku 5.2, stránka bola schopná extrahovať informácie o prehliadači do jeho odtlačku. Tento odtlačok však musel byť odoslaný na vzdialený server, ktorý tento odtlačok vyhodnotil a následne vygeneroval štatistiky odtlačku. Tieto štatistiky je možné vidieť v stĺpci *Similarity ratio*. Podobný výpis štatistík je možné pozorovať aj pri prístupe na stránku `coveryourtracks.eff.org`. V oboch prípadoch je zrejmé, že na pozadí prebehla HTTP komunikácia, o ktorej užívateľ nevie, a predsa obsahuje veľké množstvo informácií o zariadení a prehliadači.

Prístup na testovaciu stránku s modulom Fingerprinting Detection

Zapnutie modulu Fingerprinting Detection spôsobí to, že ak na navštívenej stránke prebieha extrakcia odtlačku prehliadača, všetky požiadavky od momentu detekcie budú zablokované. Výsledok prístupu na stránku `amiunique.org/fp` so zapnutím tohto modulu je znázornený na obrázku 5.3.

Javascript attributes

Attribute	Similarity ratio	Value
User agent		Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
Platform		Win32
Cookies enabled		yes
Timezone		-120
Content language		sk-SK,sk,cs,en-US,en

Obr. 5.3: Prístup na stránku `amiunique.org/fp` so zapnutým modulom Fingerprinting Detection.

Z obrázku 5.3 je zrejmé, že server zatiaľ nebol schopný poslať dáta o štatistikách získaného odtlačku. Skript síce atribúty odtlačku extrahoval a vyplnil stĺpec *Value*, na druhú stranu nebol schopný tento odtlačok odoslať na server. Z tohto dôvodu stĺpec *Similarity ratio* obsahuje obrázok načítavania, ktorý by bol v budúcnosti nahradený hodnotami štatistík. V prípade, že by dáta prehliadania neboli pri detekcií vymazané, po obnovení stránky by bol odtlačok získaný z lokálnej pamäte prehliadača a odoslaný na server. Podobné správanie je možné pozorovať aj na stránke `coveryourtracks.eff.org`, ktorej výstup je zobrazený na obrázku 5.4.

IS YOUR BROWSER:

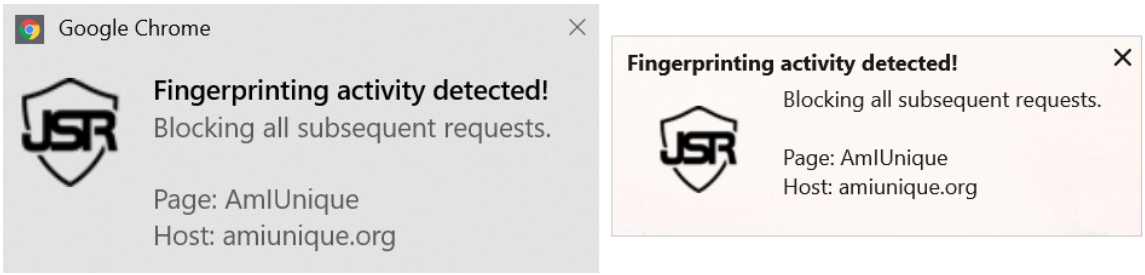
Blocking tracking ads?	No
Blocking invisible trackers?	No
Protecting you from fingerprinting?	

Obr. 5.4: Prístup na stránku `coveryourtracks.eff.org` so zapnutým modulom Fingerprinting Detection.

Zablokovanie požiadavky obsahujúcej odtlačok prehliadača je možné pozorovať aj v konzole prehliadača. Výpis po otvorení konzoly je zobrazený v hornej polovici obrázka 5.5. V dolnej polovici tohto obrázka je zobrazená časť obsahu objektu `fpDp`, ktorý zaznamenáva prístupy k obaleným prostriedkom.

Pri prístupe na obe testovacie stránky je užívateľ upozornený na skutočnosť, že modul Fingerprinting Detection detektoval podozrivú činnosť. Upozornenie má formu notifikácie, ktorá je pre jednotlivé prehliadače zobrazená na obrázku 5.6. Notifikácia obsahuje informácie o detektovanej stránke, konkrétne jej *názov* (`name`) a *meno domény* (`host`).

Obr. 5.5: Blokovanie požiadavky `POST`, ktorá obsahuje odtlačok prehliadača (hore). Časť obsahu objektu `fpDb`, v ktorom sú zaznamenané prístupy k prostriedkom po navštívení stránky `amiunique.org/fp` (dole).



Obr. 5.6: Notifikácia prehliadača Google Chrome (vľavo) a notifikácia prehliadača Mozilla Firefox (vpravo), ktorá upozorňuje na blokovanie nasledujúcich požiadaviek z dôvodu detekcie aktivity podozrivej zo získavania odtlačku prehliadača.

5.1.2 Časová a pamäťová náročnosť

Implementácia bola testovaná aj z hľadiska časovej a pamäťovej náročnosti. Za týmto účelom bola vytvorená séria testov, ktoré tieto parametre testujú. Testovanie prebiehalo v prehliadači Google Chrome s aktívnym rozšírením JavaScript Restrictor. Prehliadač beží v prostredí operačného systému Windows 10 s procesorom Ryzen 5 5600H a 16 GB pamäte RAM.

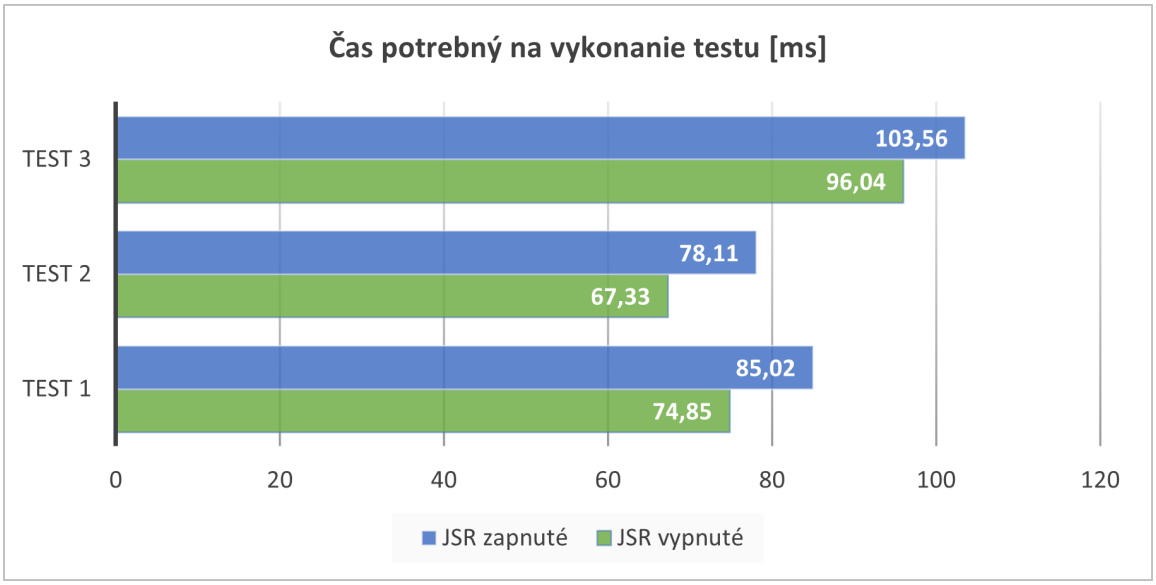
Testovanie časovej náročnosti

Testovanie časovej náročnosti testuje dopad obaľovania prostriedkov na rýchlosť pri ich používaní. Réžia spojená s odosielaním správ pri prístupe k týmto prostriedkom môže mať negatívny dopad na prehliadanie. Z tohto dôvodu je odosielanie správ jedného prostriedku obmedzené na 1000 správ za jedno navštívenie stránky.

Testovanie pozostáva z troch testov, ktoré opakovane pristupujú k vybranej množine obalených prostriedkov. Prístupy sú realizované s vypnutým a neskôr zapnutým rozšírením JavaScript Restrictor pre priame porovnanie priemerného času prístupov. Priemerný čas bol vypočítaný z výsledkov 20 nezávislých meraní. Jednotlivé testy pozostávali z nasledujúcich úloh:

- **Test 1** obsahuje 1 milión volaní metódy `Date.now()`.
- **Test 2** obsahuje 50 000 prístupov k 5 rôznym atribútom objektu `navigator`.
- **Test 3** obsahuje 1 500 vytvorení elementu `<canvas>` s následným nastavením štýlu, typu písma a textu.

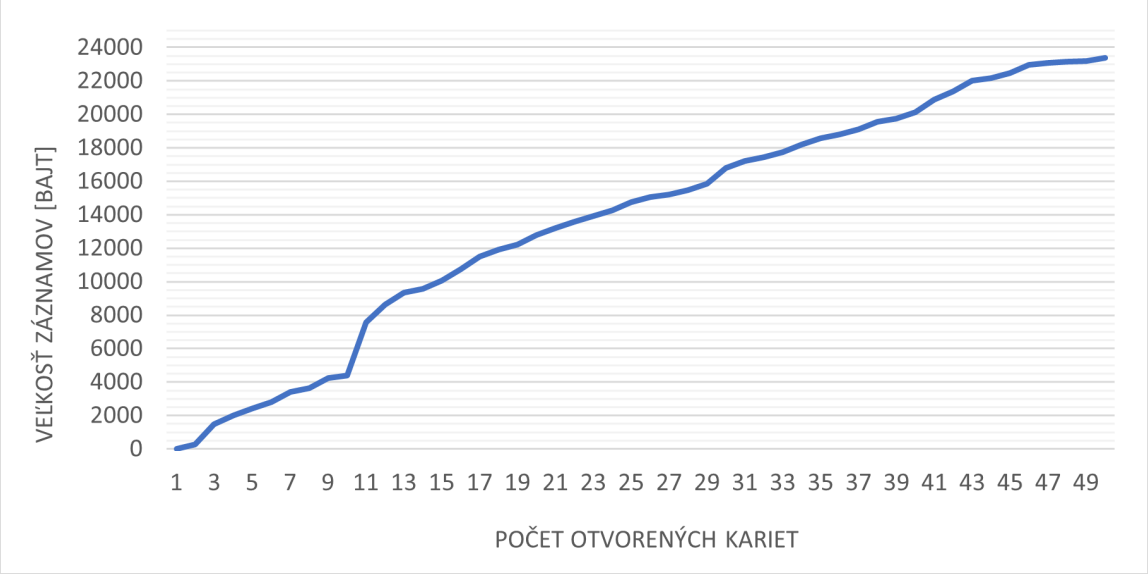
Výsledok testovania je možné vidieť v grafe na obrázku 5.7. Výsledky poukazujú na skutočnosť, že prístup k obaleným prostriedkom je v priemere o 12,5 % pomalší. Najvýraznejšie spomalenie bolo namerané vždy pri prvom behu testu, pretože vtedy bolo aktívne aj posielanie správ. Následné testy už správy neposielali, lebo hranica 1000 správ bola prekročená. Spomalenie v testoch, ktoré neposielali správy bolo zhruba 9 %.



Obr. 5.7: Výsledky testov, ktoré zobrazujú dopad obalenia prostriedkov na čas prístupu k nim.

Testovanie pamäťovej náročnosti

Testovanej pamäťovej náročnosti je založené na meraní veľkosti objektu `fpDb`, ktorý obsahuje databázu prístupov k obaleným prostriedkom. Tento objekt je nevyhnutný pre proces vyhodnocovania heuristík a celkovej detekcie získavania odtlačku prehliadača. Prístupy sú zaznamenávane pre jednotlivé prostriedky a karty prehliadača. Záznamy o prístupoch sú čistené vždy pri načítavaní nového dokumentu pre danú kartu alebo pri zatvorení karty. Tým pádom je veľkosť tohto objektu priamo závislá na počte aktuálne otvorených kariet.



Obr. 5.8: Výsledok testu veľkosti objektu `fpDb` v závislosti od počtu otvorených kariet prehliadača.

Graf na obrázku 5.8 znázorňuje veľkosť objektu `fpDb` pre rôzny počet aktuálne otvorených kariet. Pri tomto teste bola pre každú kartu otvorená jedna z prvých 50 stránok rebríčku *Alexa Global Top 500*[3]. Celková veľkosť objektu `fpDb` dosiahla 22.82 kB pri 50 otvorených kartách. Veľkosť tohto objektu narastá hlavne pri stránkach, ktoré pristupujú k veľkému množstvu obalených prostriedkov. Väčšinu výraznejších výkyvov v grafe 5.8 spôsobujú stránky, ktoré reálne vykonávajú extrakciu odtlačku prehliadača.

5.2 Testovanie detekcie na reálnych dátach

Rozšírenie JavaScript Restrictor s aktívnym modulom Fingerprinting Detection bolo testované aj na reálnej vzorke internetových stránok. Na tento účel bola využitá vzorka najpopulárnejších 50 stránok podľa rebríčka *Alexa*, podobne ako v teste pamäťovej náročnosti. Hlavným cieľom týchto testov je zhodnotiť účinnosť modulu Fingerprinting Detection z pohľadu detekcie a blokovania. Nasledujúce testy však netestujú implementáciu modulu Fingerprinting Detection ako takú, ale skôr aktuálne nastavenie vyhodnocovacích heuristík. Nastavenie heuristík vychádza z návrhu v podsekcií 3.3.2.

Problém predstavuje získanie vhodných testovacích dát, ktoré sú patrične anotované. Aby bolo možné vyhodnotiť úspešnosť detekcie, bolo nutné vybrané stránky zaradiť do skupín podľa toho či naozaj vykonávajú extrakciu odtlačku prehliadača, alebo nie. Na tento účel bol využitý zoznam domén `fingerprinting_domains`[4], na ktorých v čase štúdie [17] prebiehala extrakcia odtlačku prehliadača.

Na zvýšenie presnosti anotácie testovaných stránok bolo využité aj rozšírenie prehliadača s názvom *Don't FingerPrint Me* [5]. Toto rozšírenie umožňuje zobraziť záznamy o prístupoch k prostriedkom, ktoré sa často vyskytujú v objavených algoritmických metódach. Na základe týchto prístupov je možné vyhodnotiť, o ktorú z týchto metód sa naozaj jedná. Rozšírenie

[3] https://www.alexa.com/topsites
[4] https://github.com/uiowa-irl/FP-Inspector/blob/master/Data/fingerprinting_domains.json
[5] https://github.com/freethenation/DFPM

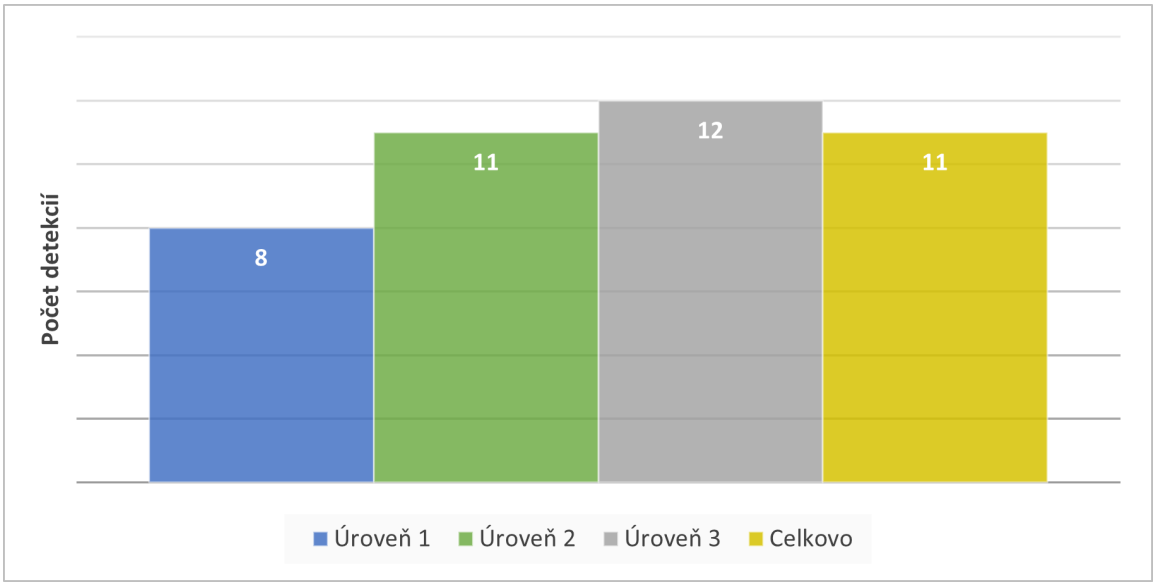
následne poukáže na podozrenia o prítomnosti týchto metód pomocou samostatnej karty v konzole prehliadača.

Výsledná anotovaná množina testovacích dát pozostáva z vybraných 50 stránok, ktoré sú ohodnotené podľa toho, či vykonávajú extrakciu odtlačku prehliadača. Stránka vykonáva odtlačok prehliadača práve vtedy, keď sa nachádza v zozname `fingerprinting_domains` a zároveň rozšírenie *Don't FingerPrint Me* poukazuje na vysoké podozrenie z tejto činnosti. Celý zoznam testovaných stránok spolu s výsledkami testovania je možné nájsť vo forme tabuliek v prílohe A.

Treba podotknúť, že oba tieto zdroje neposkytujú 100 % presnosť, čo môže výrazne ovplyvniť výsledky testov. Pri testovaní bolo zistené, že niektoré stránky už neobsahujú sledujúce skripty zo zoznamu `fingerprinting_domains`. Zároveň bolo zistené, že ani rozšírenie *Don't FingerPrint Me* nepokrýva všetky algoritmické metódy. Ako príklad je možné uviesť metódu získavania odtlačku pomocou rozhrania `AudioContext`, ktoré nebolo týmto rozšírením detektované. Naproti tomu, modul Fingerprinting Detection túto metódu validne detektoval.

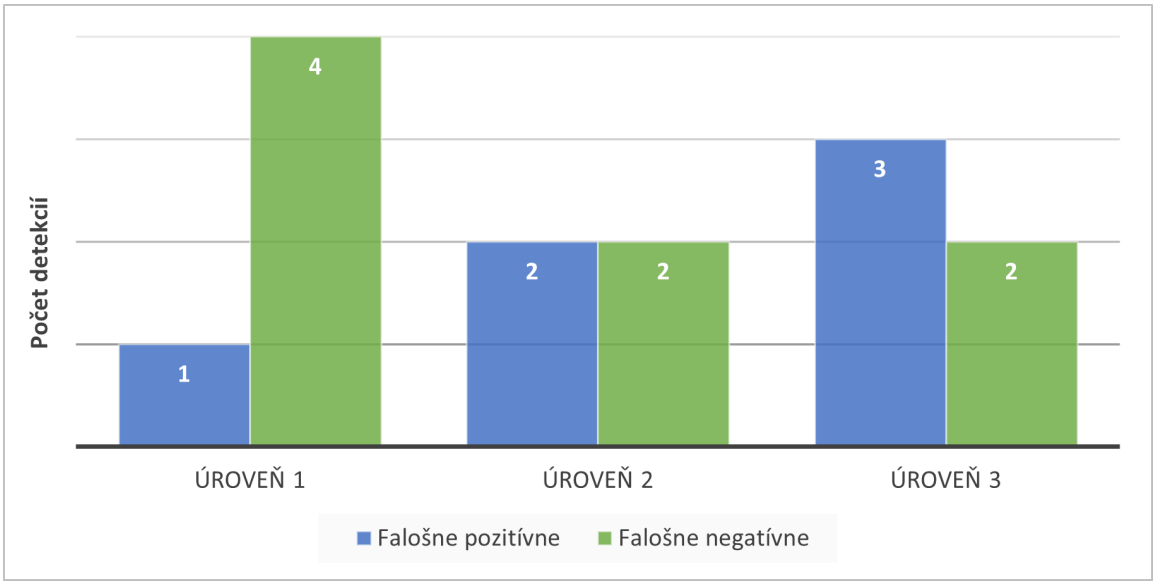
5.2.1 Úspešnosť detekcie jednotlivých úrovní heuristík

Test úspešnosti detekcie získavania odtlačku prehliadača bol realizovaný postupným prístupom na domovské stránky, ktoré sú anotované podľa predchádzajúceho postupu. Každý prístup ku stránke mal aktívnu rozdielnu úroveň ochrany rozšírenia JavaScript Restrictor. Počet detektovaných stránok pre jednotlivé úrovne je zobrazený grafom na obrázku 5.9.



Obr. 5.9: Celkový počet detekcií navštívených stránok jednotlivými úrovňami. Položka *Celkovo* označuje počet sledujúcich stránok podľa vytvorených anotácií.

Graf 5.9 poukazuje na celkové množstvo detekcií počas testovania. Čím bola úroveň ochrany vyššia, tým viac stránok bolo detektovaných. Pri testovaní je nutné brať v úvahu aj počet nesprávnych (*falošne pozitívnych*) detekcií. Okrem toho je vhodné zaznamenať aj stránky, ktoré sú označené ako sledujúce, ale modul ich nebol schopný detektovať (*falošne negatívne*). Graf týchto hodnôt pre jednotlivé úrovne je zobrazený na obrázku 5.10.



Obr. 5.10: Počet falošne pozitívnych, respektíve falošne negatívnych detekcií jednotlivých úrovní.

Najlepší výsledok dosiahli heuristiky definované pre úroveň 2. Táto úroveň predstavuje najlepší kompromis medzi falošne pozitívnymi a falošne negatívnymi detekciami. Z pohľadu úspešnosti boli heuristiky úrovne 2 schopné správne detektovať 82 % stránok, ktoré získavajú odtlačok prehliadača. Úroveň 3 bola schopná dosiahnuť rovnaký výsledok, a to 82 % správnych detekcií. Na druhej strane, úroveň 1 bola schopná správne detektovať iba 64 %, pretože kritériá heuristík sú na tejto úrovni vyššie. Výstup tohto testu treba brať s určitým nadhľadom, pretože veľkosť testovacích dát a presnosť anotácie je pomerne nízka.

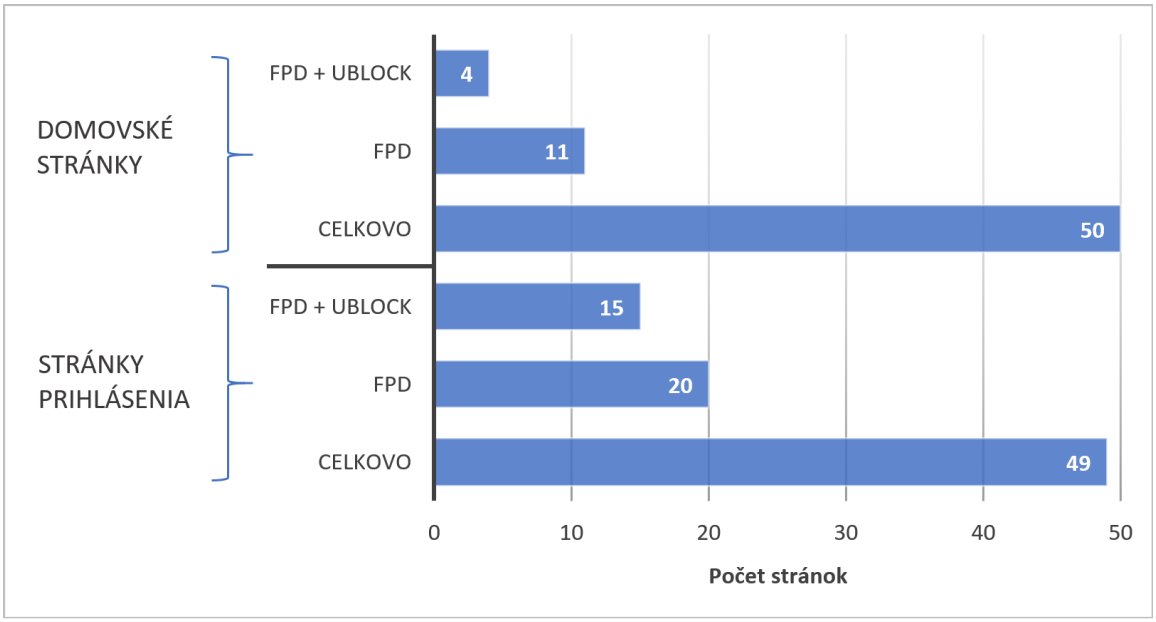
5.2.2 Testovanie detekcie s aktívnym blokovaním reklám

Stránky, na ktorých bola detektovaná extrakcia odtlačku prehliadača, môžu obsahovať skripty tretích strán, ktoré tento odtlačok získavajú na vlastné účely. Tretie strany týmto spôsobom sledujú užívateľa skrz rozličné domény a mapujú jeho aktivitu. Nástroje na blokovanie reklám využívajú zoznamy týchto tretích strán a umožňujú blokovanie skriptov, ktoré pochádzajú zo serverov tretích strán. Takto blokované skripty sa pri navštívení stránky nesťahujú, čo zamedzuje ich vykonávaniu.

Rozšírenie JavaScript Restrictor je možné používať spolu s nástrojmi na blokovanie reklám. Táto kombinácia môže posilniť ochranu proti sledovaniu. Nástroj na blokovanie reklám odfiltruje značné množstvo sledujúcich skriptov, pričom JavaScript Restrictor následne skontroluje tie skripty, ktoré odfiltrované neboli. Tento spôsob používania umožňuje znížiť negatívny efekt blokovania HTTP požiadaviek, ktorým je možné stránku po detekcií znefunkčniť. Keďže väčšina skriptov bude zablokovaná ešte pred ich vykonávaním, počet detekcií sa zníži.

Táto kombinácia bola otestovaná na rovnakých stránkach ako predchádzajúce testy. Pri testovaní bolo zistené, že mnoho stránok, ktoré nezískavajú odtlačok prehliadača na domovskej stránke, ho získavajú na stránke prihlasovania. Preto boli do tohto testu pridané aj stránky prihlasovania, ktorými je možné zvýšiť veľkosť testovacích dát. Pre tento test bola

vybratá 2. úroveň ochrany rozšírenia JavaScript Restrictor a nástroj *uBlock Origin*[6], ktorý blokuje reklamy a sledujúce stránky. Výsledky testu sú zobrazené v grafe na obrázku 5.11.



Obr. 5.11: Porovnanie počtu detekcií pri samostatnom použití rozšírenia JavaScript Restrictor s modulom Fingerprinting Detection (*FPD*) a kombinácií tohto rozšírenia s nástrojom *uBlock Origin*. Testovanie bolo vykonávané na domovských stránkach a aj na stránkach prihlasovania. Celkový počet stránok prihlasovania je nižší, pretože nie všetky stránky umožňujú prihlásenie užívateľa.

Z grafu 5.11 je zrejmé, že použitie nástroja *uBlock Origin* výrazne znížilo počet detekcií na domovských stránkach. Vďaka tomu sa zvýšila aj použiteľnosť týchto stránok, pretože ich komunikácia zo serverom nebola blokovaná. Zároveň je možné pozorovať vysoké množstvo detekcií na stránkach prihlasovania. Veľké množstvo stránok používa odtlačok prehliadača na zvýšenie bezpečnosti pri verifikácií užívateľa a jeho zariadenia. Z tohto dôvodu bolo menšie množstvo týchto stránok blokovaných nástrojom *uBlock Origin*.

5.3 Návrh vylepšení

V tejto sekcií popisujem obmedzenia aktuálnej implementácie a navrhujem možné zlepšenia. Niektoré návrhy riešia aj funkcionalitu mimo modulu Fingerprinting Detection, ktorú je možné pridať do rozšírenia JavaScript Restrictor. Jednotlivé návrhy sú popísané v nasledujúcich bodoch:

- **Pevne stanovené heuristiky** obmedzujú efektivitu detekcie. Prvá nevýhoda spočíva v tom, že heuristiky musia odrážať aktuálne trendy na poli skriptov, ktoré extrahujú odtlačok prehliadača. Prehliadače a ich prostriedky sa vyvíjajú vysokým tempom, pričom vznikajú nové možnosti na získanie identifikujúcich informácií. Heuristiky musia byť pravidelne aktualizované na to, aby boli schopné detektovať novo objavené metódy. Tento problém rieši nahradenie heuristík klasifikátorom, ktorý je založený

[6] https://github.com/gorhill/uBlock

na strojovom učení [17]. Druhý problém predstavuje „krehkosť“ nastavenia heuristík. Pevne nastavené medze jednotlivých heuristík môžu spôsobiť veľké množstvo nesprávnych klasifikácií. V aktuálnej implementácií by bolo možné definovať *hladiny podozrenia*, ktoré by boli vyjadrené váhou koreňovej skupiny. Týmto spôsobom by bolo možné reagovať rozdielnym spôsobom na rozdielne nadobudnuté váhy. Napríklad, pokiaľ by bolo nízke podozrenie, mohla by sa zablokovať iba určitá podmnožina HTTP požiadaviek.

- **Postupné získavanie odtlačku prehliadača** je aktuálna slabina implementovaného modulu. Tento modul pracuje na úrovni kariet prehliadača a premazáva záznamy prístupov po každom obnovení danej karty. Tento spôsob umožňuje zaznamenávať prístupy pri nízkej pamäťovej náročnosti. Ak by stránka bola dostatočne opatrná a atribúty odtlačku akumulovala postupne pri prehliadaní, modul nie je schopný takúto činnosť detektovať. Jedným z možných riešení je pridať detekciu na úrovni konkrétnych stránok. Táto implementácia by vyžadovala výrazne viac pamäte, pretože záznamy všetkých navštívených stránok by museli byť dočasne uchovávané.

- **Obmedzenie funkcionality** po detekcií súvisí s blokovaním HTTP požiadaviek. Aktuálne sa blokujú všetky požiadavky, ktoré by mohli slúžiť ako médium na prenos získaného odtlačku. Jedná sa o striktný, avšak účinný prístup. Na druhú stranu, tento prístup obmedzuje stránky z pohľadu funkčnosti. Navigácia v rámci stránky je umožnená, ale dynamicky načítavaný obsah je zablokovaný. Obmedzenia však úzko súvisia s implementáciou danej stránky. Jedným z riešení je už spomínané využívanie *hladín podozrenia*. Druhým možným riešením, hlavne v prípade stránok prihlásenia, je vytvorenie výnimiek na základe určitých faktorov. Napríklad, ak stránka obsahuje v adrese URL podreťazec `"login"`, pravdepodobne sa jedná o stránku prihlásenia. V tomto prípade požiadavky nebudú blokované, aby sa užívateľ mohol bez problémov prihlásiť.

- **Podpora užívateľsky voliteľných heuristík** je funkcionalita, ktorá by umožňovala definíciu heuristík pomocou užívateľského prostredia. Pokročilý užívateľ by si tak sám vedel stanoviť kritériá a prostriedky, ktoré chce zaznamenávať a vyhodnocovať pri detekcií. Jedná sa však o zložitý proces, pretože určenie týchto heuristík nie je jasne dané a je na zváženie, či koncový užívateľ bude ich definície schopný. Naproti tomu, rozšírenie JavaScript Restrictor podporuje definíciu užívateľsky konfigurovateľných úrovní ochrany, ktoré aktuálna implementácia modulu ignoruje. Do budúcna je vhodné umožniť špecifikovať detekciu získavania odtlačku prehliadača aj pre tieto úrovne.

- **Pasívny odtlačok prehliadača** je možné získať aj napriek implementovanému modulu. Tento modul totižto rieši len odtlačok získateľný pomocou rozhraní prehliadača a jazyka JavaScript. Jedno z možných rozšírení na ochranu pred získavaním odtlačku prehliadača môže byť modifikácia informácií, ktoré tvoria súčasť pasívneho odtlačku.

Kapitola 6

Záver

Súčasný stav prostredia internetu predstavuje vážne riziko z pohľadu súkromia. Rozšírenosť internetového sledovania stále rastie a neustále pribúdajú spôsoby, ako identifikovať jednotlivých užívateľov internetu. Medzi čoraz používanejšie spôsoby patrí aj využívanie odtlačku prehliadača, ktorý dopĺňa a niekedy aj nahradzuje stavové identifikátory. Vysoká miera jedinečnosti odtlačku prehliadača je založená na vysokej rozmanitosti zariadení, ktoré majú prístup na internet. Táto práca sa venovala vývoju mechanizmu, ktorý umožní sledovanie pomocou odtlačku prehliadača obmedziť.

Kapitola 2 sa venovala bezstavovému sledovaniu odtlačkom prehliadača. V tejto kapitole som zhrnul aktuálny stav tejto problematiky, poukázal na použiteľnosť a reálne nasadenie tohto typu sledovania. Ďalej som predstavil príklad reálneho odtlačku prehliadača, ktorý bol vytvorený aktuálne dostupným nástrojom. Vďaka tomu som mohol poukázať na základnú štruktúru odtlačku prehliadača a vytvoriť prehľad častých atribútov odtlačku. V zápätí som zhrnul možnosti obrany proti tomuto typu sledovania.

V nasledujúcej kapitole 3 som predstavil návrh mechanizmu, ktorý vychádza z analýzy existujúcich nástrojov na detekciu získavania odtlačku prehliadača. Vďaka detekcií je následne možné zabrániť odoslaniu citlivých dát na server sledujúcej strany. Keďže mechanizmus má byť implementovaný vo forme rozšírenia prehliadača, kapitola 3 sa taktiež venuje tvorbe rozšírení pomocou *WebExtensions API*.

Kapitola 4 popisuje postup implementácie modulu *Fingerprinting Detection*, ktorý tvorí súčasť rozšírenia *JavaScript Restrictor*. Hlavným cieľom tohto modulu je vedieť detektovať podozrivú činnosť na základe prístupov k rozličným rozhraniam, ktoré poskytuje internetový prehliadač. Kapitola 4 popisuje integráciu tohto modulu v rámci rozšírenia *JavaScript Restrictor* a poukazuje na zaujímavé implementačné detaily. Implementácia je rozdelená na logické celky, ktoré sú zodpovedné za jednotlivé kroky detekcie a blokovania. V tejto kapitole taktiež popisujem štruktúru konfiguračných súborov, ktoré obsahujú vyhodnocovacie heuristiky používané pri detekcií.

V kapitole 5 demonštrujem použitie vytvoreného modulu. Funkcionalita je ukázaná na nástrojoch, ktoré sú určené na extrakciu odtlačku prehliadača. Ďalej v tejto kapitole testujem dopad tohto rozšírenia na rýchlosť pri prístupe k prostriedkom, ktoré boli obalené za účelom ich zaznamenávania. Testy poukázali na zanedbateľné hodnoty časovej a priestorovej náročnosti implementovaného modulu. Testovanie prebiehalo aj na bežných stránkach, kde bola vyhodnotená úspešnosť detekcie navrhnutými heuristikami. Modul bol schopný detektovať 82 % stránok, ktoré získavali odtlačok prehliadača. Toto číslo je výrazne ovplyvnené nízkou presnosťou anotácie testovaných stránok. Pri podrobnejšom prieskume testovaných stránok som zistil, že modul *Fingerprinting Detection* reagoval oprávnene a všetky detek-

tované stránky obsahovali podozrivé skripty. Preto predpokladám, že úspešnosť detekcie je omnoho vyššia ako spomenutá hodnota. V rámci kapitoly 5 som poukázal aj na použitie kombinácie rozšírenia JavaScript Restrictor a nástroja na blokovanie reklám, čo umožnilo znížiť počet detekcií a zvýšiť použiteľnosť stránok. Koniec tejto kapitoly sa venuje popisu obmedzení a možných zlepšení rozšírenia.

Na základe testovania sa implementácia modulu *Fingerprinting Detection* ukázala byť funkčná a použiteľná v reálnom nasadení. Dokonca aj navrhnuté heuristiky predstavujú solídny základ, ktorý dokáže s vysokou presnosťou detektovať získavanie odtlačku prehliadača. Verím, že tento modul poskytne zlepšenie súkromia pri prehliadaní internetu s rozšírením JavaScript Restrictor.

Literatúra

[1] Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A. et al. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. *Proceedings of the ACM Conference on Computer and Communications Security*. November 2014, s. 674–689. DOI: 10.1145/2660267.2660347.

[2] Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gurses, S. et al. FPDetective: Dusting the web for fingerprinters. In:. November 2013, s. 1129–1140. DOI: 10.1145/2508859.2516674.

[3] Al Fannah, N., Li, W. a Mitchell, C. Beyond Cookie Monster Amnesia: Real World Persistent Online Tracking. In:. September 2018, s. 481–501. DOI: 10.1007/978-3-319-99136-8_26. ISBN 978-3-319-99135-1.

[4] Bray, T. *The JavaScript Object Notation (JSON) Data Interchange Format* [Internet Requests for Comments]. STD 90. RFC Editor, December 2017.

[5] Bujlow, T., Carela-Español, V., Solé-Pareta, J. a Barlet-Ros, P. A Survey on Web Tracking: Mechanisms, Implications, and Defenses. *Proceedings of the IEEE*. 2017, zv. 105, č. 8, s. 1476–1510. DOI: 10.1109/JPROC.2016.2637878.

[6] Cao, Y., Li, S. a Wijmans, E. (Cross-)Browser Fingerprinting via OS and Hardware Level Features. In:. Január 2017. DOI: 10.14722/ndss.2017.23152.

[7] Doty, N. *Mitigating Browser Fingerprinting in Web*. Editor's Draft. W3C, marec 2020. Dostupné z: https://w3c.github.io/fingerprinting-guidance/.

[8] Eckersley, P. How Unique is Your Web Browser? In:. Január 2010, sv. 6205, s. 1–18. DOI: 10.1007/978-3-642-14527-8_1. ISBN 978-3-642-14526-1.

[9] Englehardt, S., Reisman, D., Eubank, C., Zimmerman, P., Mayer, J. et al. Cookies that give you away: The surveillance implications of web tracking. *Proceedings of the 24th International Conference on World Wide Web*. Január 2015, s. 289–299.

[10] Englehardt, S. a Narayanan, A. Online Tracking: A 1-million-site Measurement and Analysis. In:. Október 2016, s. 1388–1401. DOI: 10.1145/2976749.2978313.

[11] Fielding, R. T. a Singer, D. *Tracking Preference Expression (DNT)*. Working Group Note. W3C, január 2019. Dostupné z: https://www.w3.org/TR/tracking-dnt/.

[12] FingerprintJS. *Fingerprintjs* [online]. [cit. 2020-01-01]. Dostupné z: https://github.com/fingerprintjs/fingerprintjs.

[13] Google Docs. *Chrome.browserAction* [online]. [cit. 2020-01-09]. Dostupné z: https://developer.chrome.com/docs/extensions/reference/browserAction/.

[14] Google Docs. *Overview of Manifest V3* [online]. [cit. 2020-01-11]. Dostupné z: https://developer.chrome.com/docs/extensions/mv3/intro/mv3-overview/.

[15] Gulyas, G. G., Some, D. F., Bielova, N. a Castelluccia, C. To Extend or Not to Extend: On the Uniqueness of Browser Extensions and Web Logins. In: *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*. New York, NY, USA: Association for Computing Machinery, 2018, s. 14–27. WPES'18. DOI: 10.1145/3267323.3268959. ISBN 9781450359894.

[16] Gómez Boix, A., Laperdrix, P. a Baudry, B. Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. In:. Apríl 2018, s. 309–318. DOI: 10.1145/3178876.3186097. ISBN 978-1-4503-5639-8.

[17] Iqbal, U., Englehardt, S. a Shafiq, Z. Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors. In: *2021 IEEE Symposium on Security and Privacy*. 2021.

[18] Kamkar, S. *Evercookie - virtually irrevocable persistent cookies* [online]. [cit. 2020-12-21]. Dostupné z: http://samy.pl/evercookie/.

[19] Khattak, S., Fifield, D., Afroz, S., Javed, M., Sundaresan, S. et al. Do You See What I See? Differential Treatment of Anonymous Users. In:. Február 2016. DOI: 10.14722/ndss.2016.23342.

[20] Laperdrix, P., Rudametkin, W. a Baudry, B. Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints. In: *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, s. 878–894. DOI: 10.1109/SP.2016.57.

[21] Laperdrix, P., Bielova, N., Baudry, B. a Avoine, G. Browser Fingerprinting: A Survey. *ACM Trans. Web*. New York, NY, USA: Association for Computing Machinery. apríl 2020, zv. 14, č. 2. DOI: 10.1145/3386040. ISSN 1559-1131.

[22] Mayer, J. R. *Any person... a pamphleteer": Internet Anonymity in the Age of Web 2.0*. USA, 2009. Diplomová práca. Undergraduate Senior Thesis, Princeton University. Dostupné z: https://jonathanmayer.org/publications/thesis09.pdf.

[23] MDN contributors. *Anatomy of an extension* [online]. [cit. 2020-12-13]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Anatomy_of_a_WebExtension.

[24] MDN contributors. *Browser support for JavaScript APIs* [online]. [cit. 2020-01-09]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Browser_support_for_JavaScript_APIs.

[25] MDN contributors. *Chrome incompatibilities* [online]. [cit. 2020-01-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Chrome_incompatibilities.

[26] MDN CONTRIBUTORS. *Javascript* [online]. [cit. 2021-05-02]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty.

[27] MDN CONTRIBUTORS. *NetworkInformation* [online]. [cit. 2021-01-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/NetworkInformation.

[28] MDN CONTRIBUTORS. *WebRequest* [online]. [cit. 2020-01-09]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest.

[29] MDN CONTRIBUTORS. *What are extensions?* [online]. [cit. 2020-01-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/What_are_WebExtensions.

[30] MOWERY, K. a SHACHAM, H. Pixel Perfect: Fingerprinting Canvas in HTML5. In: FREDRIKSON, M., ed. *Proceedings of W2SP 2012*. Máj 2012.

[31] NAKIBLY, G., SHELEF, G. a YUDILEVICH, S. *Hardware Fingerprinting Using HTML5*. 2015.

[32] NIKIFORAKIS, N., KAPRAVELOS, A., JOOSEN, W., KRUEGEL, C., PIESSENS, F. et al. Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting. In: *2013 IEEE Symposium on Security and Privacy*. 2013, s. 541–555. DOI: 10.1109/SP.2013.43.

[33] NORTE, J. C. *Advanced Tor Browser Fingerprinting* [online]. [cit. 2020-01-08]. Dostupné z: http://jcarlosnorte.com/security/2016/03/06/advanced-tor-browser-fingerprinting.html.

[34] PERRY, M., CLARK, E., MURDOCH, S. a KOPPEN, G. *The Design and Implementation of the Tor Browser* [online]. [cit. 2020-12-25]. Dostupné z: https://2019.www.torproject.org/projects/torbrowser/design/.

[35] PIETRASZAK, M. *Browser Extensions*. Draft Community Group Report. W3C, január 2020. Dostupné z: https://browserext.github.io/browserext/.

[36] POHNER, P. *Detekce podezřelých síťových požadavků webových stránek*. Brno, CZ, 2020. Diplomová práca. Vysoké učení technické v Brně, Fakulta informačních technologií.

[37] POLČÁK, L. *The configuration of the extension* [online]. [cit. 2020-01-10]. Dostupné z: https://polcak.github.io/jsrestrictor/levels.html.

[38] POLČÁK, L. *JavaScript Restrictor* [online]. [cit. 2020-01-10]. Dostupné z: https://polcak.github.io/jsrestrictor/.

[39] SANCHEZ ROLA, I., SANTOS, I. a BALZAROTTI, D. Clock Around the Clock: Time-Based Device Fingerprinting. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2018, s. 1502–1514. CCS '18. DOI: 10.1145/3243734.3243796. ISBN 9781450356930.

[40] Schwarz, M., Lackner, F. a Gruss, D. JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits. In: *NDSS*. 2019.

[41] Sjösten, A., Van Acker, S. a Sabelfeld, A. Discovering Browser Extensions via Web Accessible Resources. In: *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. New York, NY, USA: Association for Computing Machinery, 2017, s. 329–336. CODASPY '17. DOI: 10.1145/3029806.3029820. ISBN 9781450345231.

[42] Snyder, P. E. *Improving Web Privacy And Security with a Cost-Benefit Analysis of the Web API*. University of Illinois at Chicago, Apr 2018.

[43] Starov, O. a Nikiforakis, N. XHOUND: Quantifying the Fingerprintability of Browser Extensions. In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, s. 941–956. DOI: 10.1109/SP.2017.18.

[44] Vastel, A., Laperdrix, P., Rudametkin, W. a Rouvoy, R. Fp-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies. In: *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, August 2018, s. 135–150. ISBN 978-1-939133-04-5.

[45] Červinka, Z. *Rozšíření pro webový prohlížeč zaměřené na ochranu soukromí*. Brno, CZ, 2018. Diplomová práca. Vysoké učení technické v Brně, Fakulta informačních technologií.

Príloha A

Zoznam testovaných stránok

Tabuľky A.1 a A.2 obsahujú zoznam internetových stránok, ktoré boli využité pri testovaní v kapitole 5. Výsledky testovania sú rozdelené podľa typu navštívenej stránky. Skratka *DS* predstavuje *domovské stránky* a skratka *SP* predstavuje *stránky prihlásenia*.

Za typom stránky je označenie konfigurácie prístupu na stránku. Písmeno *A* sú anotácie, *číselná hodnota* označuje úroveň ochrany rozšírenia JSR a písmeno *U* predstavuje použitie nástroja *uBlock Origin*.

	DS A	DS 1	DS 2	DS 3	DS 2+U	SP 2	SP 2+U
google.com	0	0	0	0	0	0	0
youtube.com	0	0	0	0	0	0	0
tmall.com	1	1	1	1	0	1	0
baidu.com	0	0	1	1	1	1	1
qq.com	0	0	0	0	0	0	0
sohu.com	1	1	1	1	0	0	0
facebook.com	0	0	0	0	0	0	0
taobao.com	1	1	1	1	0	1	0
jd.com	1	1	1	1	0	1	0
360.cn	0	0	0	0	0	0	0
amazon.com	0	0	0	0	0	1	1
yahoo.com	1	0	0	0	0	1	1
wikipedia.org	0	0	0	0	0	0	0
zoom.us	0	0	0	0	0	1	1
weibo.com	0	0	0	0	0	0	0
xinhuanet.com	0	0	0	0	0	0	0
sina.com.cn	0	0	0	1	0	0	0
live.com	0	0	0	0	0	0	0
reddit.com	0	0	0	0	0	0	0
office.com	0	0	0	0	0	0	0

Tabuľka A.1: Prvých 20 internetových stránok, ktoré boli použité pri testovaní. Hodnota 1 označuje stránky, na ktorých bolo detektované získavanie odtlačku prehliadača podľa zvolenej konfigurácie. Hodnota 0 označuje stránky, na ktorých toto správanie nebolo detektované.

	DS A	DS 1	DS 2	DS 3	DS 2+U	SP 2	SP 2+U
microsoft.com	0	0	0	0	0	0	0
netflix.com	0	0	0	0	0	0	0
instagram.com	0	0	0	0	0	0	0
csdn.net	0	0	0	0	0	1	1
google.com.hk	0	0	0	0	0	0	0
zhanqi.tv	0	0	0	0	0	1	1
vk.com	1	1	1	1	1	1	1
panda.tv	0	0	0	0	0		
shopify.com	0	0	0	0	0	0	0
alipay.com	0	0	0	0	0	1	1
bing.com	0	0	0	0	0	0	0
yahoo.co.jp	0	0	0	0	0	0	0
naver.com	0	0	0	0	0	1	1
okezone.com	1	0	1	1	0	1	0
binance.com	1	1	1	1	1	1	1
twitter.com	0	0	0	0	0	0	0
twitch.tv	0	0	0	0	0	0	0
bongacams.com	0	0	0	0	0	0	0
adobe.com	1	0	1	1	0	0	0
ebay.com	1	0	0	0	0	1	0
yy.com	0	1	1	1	1	1	1
tianya.cn	0	0	0	0	0	0	0
amazon.in	0	0	0	0	0	1	1
amazon.co.jp	0	0	0	0	0	1	1
linkedin.com	0	0	0	0	0	1	1
aliexpress.com	1	1	1	1	0	1	1
huanqiu.com	0	0	0	0	0	0	0
aparat.com	0	0	0	0	0	0	0
17ok.com	0	0	0	0	0	0	0
apple.com	0	0	0	0	0	0	0

Tabuľka A.2: Zvyšných 30 internetových stránok, ktoré boli použité pri testovaní. Hodnota 1 označuje stránky, na ktorých bolo detektované získavanie odtlačku prehliadača podľa zvolenej konfigurácie. Hodnota 0 označuje stránky, na ktorých toto správanie nebolo detektované.