FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Fingerprinting prohlížeče - techniky a obrana
Student:	Bc. Samuel Hanák
Vedoucí:	Ing. Josef Kokeš
Studijní program:	Informatika
Studijní obor:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

1) Seznamte se s technikami fingerprintingu webového prohlížeče. Zaměřte se na vývoj oblasti v posledních třech letech.
2) Analyzujte úpravy v moderních verzích prohlížečů, kterými je dopad fingerprintingu omezován.
3) Prostudujte možnosti a implementaci nástroje Privoxy (https://www.privoxy.org). Proveďte rešerši jeho stávajících možností v obraně vůči fingerprintingu.
4) Navrhněte technické úpravy Privoxy pro rozšíření jeho antifingerprintingových opatření.
5) Implementujte navržené úpravy, otestujte jejich účinnost a vliv na výkon Privoxy i bezpečnost uživatele.
6) Diskutujte své výsledky.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 10. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Fingerprinting prohlížeče – techniky a obrana

Bc. Samuel Hanák

Katedra informační bezpečnosti
Vedoucí práce: Ing. Josef Kokeš

27. května 2020

Poděkování

Nejhlubší dík patří mé rodině a přítelkyni za to, že při mně po celou tu dobu stáli. Za svědomité a trpělivé vedení mé práce děkuji Ing. Josefu Kokešovi.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným zpusobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 27. května 2020 . . . . . .

Odkaz na tuto práci

Hanák, Samuel. *Fingerprinting prohlížeče – techniky a obrana.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

V práci je představeno množství technik fingerprintingu prohlížeče a způsob jejich fungování. Proti některým z nich je navržena a vysvětlena možná obrana. V návaznosti na předchozí práci je poté popsán způsob implementace navržených obran v proxy serveru Privoxy a jejich testování pomocí dostupných nástrojů.

Klíčová slova fingerprinting, webový prohlížeč, sledování uživatelů, Privoxy

Abstract

This thesis presents a number of browser fingerprinting techniques and explains how they work. Possible defenses against some of them are proposed and explained. Building on the previous work, the implementation of the proposed defenses in the Privoxy proxy server and their testing using available tools is then described.

Keywords fingerprinting, web browser, user tracking, Privoxy

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Soukromí a anonymita uživatelů na internetu je palčivé a z mnohých stran zkoumané téma. S rozšiřující se škálou aktivit, které uživatelé na internetu běžně provádějí, roste i riziko spojené s monitorováním těchto aktivit a ukládáním citlivých dat. Kauzy týkající se úniku soukromých uživatelských dat (jako například kauza shromažďování dat uživatelů Facebooku konzultační firmou Cambridge Analytica, která byla značně medializovaná začátkem roku 2018 [1]) bývají v poslední době sledované médii a potažmo i širokou veřejností. Existuje však i způsob shromažďování uživatelských dat, který se v různé míře zkoumá už delší dobu, stále se rozvíjí a navíc poskytuje provozovatelům webových služeb velké množství údajů o jejich uživatelích, kteří o tom mnohdy vůbec neví.

Lou Montulli, zaměstnanec Netscape Communications, navrhl pro účely rozpoznávání konkrétních uživatelů (respektive jejich jednotlivých relací – tzv. „sessions“) roku 1994 HTTP cookies [2]. Krátce po jejich vzniku byly cookies přijaty všemi oblíbenými prohlížeči a významným způsobem pozitivně ovlivnily uživatelský komfort a potenciál webových služeb, neboť v podstatě položily základ moderní autentizace a stavovosti webových aplikací. Přesto však brzy přišlo vystřízlivění, když si široká veřejnost (díky značné medializaci) uvědomila, že identifikace uživatele pomocí cookies lze využít nejen k přihlašování, ale také k usnadnění sledování uživatelské aktivity, které bylo velkou hrozbou pro soukromí a anonymitu. Zvláštní obavy tehdy vyvolaly tzv. cookies třetích stran, které byly (a stále jsou) hojně využívané ke sledování uživatelů, nejčastěji pro účely spojené s reklamou.

Cookies ušly od té doby dlouhou cestu a kromě určitých regulací (především Směrnice o soukromí a elektronických komunikacích [3]) došlo hlavně k tomu, že prohlížeče poskytují jednoduché nástroje, pomocí nichž lze s cookies pracovat (anonymní režim, možnost selektivně mazat cookies apod.), a uživatelé se tyto nástroje naučili používat. Tím se sledování uživatelů pomocí cookies dostalo do velké míry znovu pod kontrolu uživatelů. Přirozeně

však přišla i snaha využít ke sledování uživatelské aktivity dalších nástrojů, tentokrát však takových, které k tomu nebyly zamýšleny.

Fingerprinting je „*technika umožňující sledování a identifikaci uživatelů na základě nastavení jejich prohlížeče a použitého zařízení*“ (vlastní překlad) [4]. Moderní webové prohlížeče poskytují navštíveným serverům celou řadu informací, které lze k identifikaci využít, některé z nich v podstatě nedobrovolně, jiné zcela záměrně. Důvodem pro sdílení těchto dat ze strany prohlížeče je rozšiřování pole poskytovaných služeb a snaha zpříjemnit a zjednodušit uživateli jejich používání. Cílem této práce je prostřednictvím rozšíření proxy serveru Privoxy [5] umožnit uživatelům o něco větší kontrolu nad tím, jaká data jsou prostřednictvím prohlížeče poskytována provozovatelům webových služeb.

Fingerprintingových technik je celá řada a jejich kombinací lze získat výjimečný a někdy i zcela unikátní „otisk“ zařízení, které uživatel používá. Rozboru a důkladnému vysvětlení mnoha těchto technik se věnuje diplomová práce Ing. Adama Prášila [6]. Ing. Jiří Sixta ve své práci implementoval obranu proti vybraným fingerprintingovým technikám jako rozšíření do proxy serveru Privoxy [7]. V následujících stránkách budu navazovat na tyto práce. V kapitole Fingerprintingové techniky a jejich vlastnosti shrnu fingerprintingové techniky. Zvláštní pozornost při tom budu věnovat technikám, které budou součástí mého rozšíření Privoxy. V kapitole Obranné techniky a nástroje rozeberu obranné strategie proti fingerprintingu a analyzuji současný stav implementace těchto strategií ve skutečných nástrojích. V kapitole Analýza obran navržených pro Privoxy shrnu fingerprintingové techniky implementované v Privoxy a výstupy práce [7], na kterou navazuji. V poslední kapitole Návrh a implementace podrobněji rozvedu, jakým způsobem jsou realizovány obrany proti fingerprintingu, které jsem se rozhodl implementovat.

Kapitola 1

Fingerprintingové techniky a jejich vlastnosti

Statistika [8] uvádí, že se na internetu pohybuje 4,39 miliard uživatelů, přičemž v průměru tráví lidé na internetu více než šest a půl hodiny denně. To pochopitelně vytváří obrovský prostor pro monitorování uživatelských aktivit. Ke sledování uživatelů se webové služby uchylují z následujících důvodů:

- Jedním z nejčastějších a nejpřirozenějších důvodů pro fingerprinting v průmyslu je **personalizace reklamy**. Monitorování aktivit uživatele (nejlépe napříč různými webovými stránkami) je velmi efektivním nástrojem pro zvýšení relevance reklam [9], která pochopitelně vede k větším ziskům – typicky informace o tom, že si uživatel prohlížel nějaké zboží, je dobrým vodítkem pro to, že bude výhodné mu stejné nebo podobné zboží nabízet v budoucnosti znovu prostřednictvím reklamy. Aby marketingové společnosti byly schopné detekovat, že se jedná o stejného uživatele (což je pro takový proces nezbytné), mohou využít (a využívají) právě fingerprinting.

- Podobně jako pro zlepšení relevance reklamy lze využít fingerprinting pro **vylepšování provozované služby**. Pomocí otisku prohlížeče lze návštěvníka stránek identifikovat napříč vícerými návštěvami a nabízet mu obsah, který je pro něj zajímavý (na základě informace o obsahu, který si prohlížel dříve).

- Zcela legitimní, i když méně častý motiv pro fingerprinting je **odhalení podvodu** nebo pokusu o zneužití cizího uživatelského účtu ve webové službě [10]. Podle otisku prohlížeče, který provozovatel služby získá a přiřadí ke konkrétnímu uživateli, lze detekovat, zda jsou nové pokusy o přihlášení (pravděpodobně) legitimní. Je běžnou praxí, že služba, která detekuje významný rozdíl mezi běžným otiskem prohlížeče uživatele a

novým otiskem získaným při přihlášení, informuje uživatele o této události. Tomu je tak umožněno buď reagovat na událost jako na bezpečnostní incident (a například změnit své heslo nebo zablokovat účet), nebo naopak potvrdit oprávněnost přihlášení a přidat nový otisk mezi „důvěryhodné".

– Méně známou motivaci pro fingerprinting představuje **detekce botů** [11]. Historicky se stránky rozhodovaly o tom, zda konkrétní spojení nebo požadavek povolí, na základě veřejné IP adresy a její reputace. Tím se snažily redukovat nebezpečí spojené s externími připojeními. Fingerprinting prohlížeče umožňuje shromáždit daleko větší množství informací, které mohou sloužit k rozpoznání například známek automatizace.

– Ashouri a Asadian [12] uvádí, že některé stránky (například Piano Media [13]) využívají fingerprinting, aby zajistily, že **uživatelé nepřistupují k prémiovému obsahu**. Provozovatel stránek například chce mít přehled o tom, jak velké množství obsahu si konkrétní uživatel prohlédl, a od určité hranice požadovat registraci nebo platbu. K tomu mu může pomoci právě fingerprinting.

– Zajímavé – i když v současnosti spíše okrajové – využití fingerprintingu je **rozpoznání ukradení uživatelské relace** (session) na webové stránce. Framework SHPF (Session Hijacking Prevention Framework) [14] využívá otisku prohlížeče svých uživatelů k tomu, aby detekoval, zda cookie náležící ke konkrétnímu sezení náhle nezměnila prohlížeč. Takové chování by totiž potenciálně mohlo indikovat pokus o uloupení sezení uživatele (například technikou cross-site scripting).

V této kapitole představíme několik fingerprintingových technik. Zvláštní pozornost budeme věnovat třem skupinám:

– Technikám, které v poslední době prodělaly významnější změny (typicky změnu podpory ze strany prohlížečů),

– technikám, pro které byl v rámci této práce implementován nějaký druh obrany jako rozšíření do Privoxy (tyto obranné techniky budou později představeny v Návrh a implementace),

– technikám, které nebyly představeny v rámci prací, na které navazujeme [7] [6] (typicky proto, že ještě neexistovaly, nebo jejich využívání není z nějakého důvodu v praxi příliš populární, přestože představují zajímavou oblast zkoumání).

1.1 Mobilní zařízení

Čas strávený na internetu tvoří podle statistiky [8] zhruba ze 48 % užívání mobilních zařízení. Uživatelé chytrých telefonů a tabletů jsou tak velmi zajímavým cílem sledování. Jak uvádí Sixta [7], velké množství fingerprintingových technik nelze na mobilním zařízení použít v plné míře – například velmi rozšířená a jinak účinná technika fingerprintingu pomocí seznamu fontů není v případě mobilních zařízení příliš užitečná, neboť sada písem nainstalovaných na konkrétním zařízení je typicky dlouhodobě neměnná a velmi podobná s ostatními zařízeními. Podobně využívání zásuvných modulů v prohlížeči, které je webovou službou detekovatelné, není u mobilních prohlížečů typické, a proto ani tato metoda nebude mít dostatečný efekt.

Na druhou stranu Adobe Flash, který na běžných mobilních zařízeních už delší dobu není podporován, byl delší čas využíván jako účinný fingerprintingový nástroj právě pro desktop, a jeho podpora bude letošním rokem ukončena [15]. Desktopové prohlížeče postupně snižovaly podporu pro Flash Player a snažily se své uživatele od jeho používání odradit už nějakou dobu. Google Chrome od verze 76 v roce 2019 zablokoval Flash Player s tím, že uživatelé ho mohou povolit pouze prostřednictvím nastavení a potom musí dát explicitní souhlas se spuštěním pro každou stránku, kdykoliv se prohlížeč restartuje [16]. Podobně i Mozilla Firefox počínaje verzí 69 přestala ve výchozím stavu podporovat Adobe Flash [17]. Přestože se následkem těchto změn Adobe Flash už delší dobu vytrácí ze standardní uživatelské činnosti, fakt, že bude zcela odstraněn ze všech zařízení, je důležitým krokem ke zlepšení podmínek uživatelů na internetu, pokud jde o soukromí a bezpečnost. Jistý dopad však tato skutečnost bude mít i na rozdíl sledovatelnosti mobilních a desktopových zařízení, protože Flash Player dnes lze použít k fingerprintingu prakticky pouze desktopových zařízení.

Oliver [18] uvádí, že šance získat unikátní otisk je velmi podobná pro desktopové i mobilní zařízení – v experimentu, který byl proveden nad datasetem poskytnutým od *Am I Unique?*, bylo 91 % desktopových zařízení identifikováno jako unikátní (tj. v použitém datasetu nebylo nalezeno jiné zařízení se stejným otiskem). Podíl unikátních otisků pro mobilní zařízení činil 90 %. Podobně i výsledky pro jedinečnost mobilních zařízení používajících Android a iOS jsou srovnatelné – pro Android bylo v datasetu nalezeno více než 75 % jedinečných zařízení, pro iOS zhruba 71 %.

1.2 Objekt „navigator“ a JavaScriptové API

Rozhraní `Navigator` dostupné prostřednictvím JavaScriptu obsahuje velké množství obecných informací o používaném prohlížeči. Obrana proti využití objektu `navigator` pro tvorbu otisku prohlížeče tvoří podstatnou část rozšíření Privoxy v dosavadní práci [7]. Velké množství dat, které `navigator`

zprostředkovává, pochází z API, jež bylo v nedávné době rozšířeno a prošlo podstatnými změnami, proto se na některé jeho možnosti podíváme blíže, okomentujeme je v souvislosti s fingerprintingem a představíme úpravy, kterými prošly v poslední době.

1.2.1 Battery Status API

Battery Status API poskytuje informace o úrovni nabití baterie a poskytuje upozornění na změny úrovně nabití baterie a připojení nebo odpojení zařízení do nabíječky [19]. Cílem zavedení tohoto API bylo:

- Umožnit vývojářům webových stránek dynamicky upravovat výpočetní náročnost spouštěného kódu (a v důsledku tedy spotřebu elektrické energie) na základě informací o nabití zařízení.
- Umožnit „nouzové“ uložení stavu aplikace, pokud se baterie blíží úplnému vybití, aby se zabránilo ztrátě dat.

Většina prohlížečů implementovala toto API a i v současné době je podpora aktivní a rozšířená (aktuálně je podporován prohlížeči Google Chrome, Microsoft Edge, Opera, Android Browser a dalšími [20]).

Roku 2015 nicméně vyšla práce zabývající se problémem sledování uživatelů za využití právě Battery Status API [21]. Práce ukazuje, že detekce úrovně nabití baterie s vysokou přesností může vést k odhalení kapacity baterie uživatele. Dále pak uvádí, že informace o baterii lze využít jako krátkodobý identifikátor zařízení napříč více webovými stránkami, i k rozpoznání zařízení, která mají jinak velmi podobné vlastnosti a využívají maškarádu (NAT 1:N), například v korporátním prostředí.

Práce [21] dále ukazuje, že Battery Status API je využíváno téměř výhradně pro účely fingerprintingu. Zajímavé přitom je, že takový rozsah pole pro sledování uživatele autoři API zřejmě nepředvídali, neboť v původním návrhu W3C [22] stojí, že „*informace, které API poskytuje, mají minimální dopad na soukromí nebo fingerprinting, a proto je vystaveno bez nutnosti udělit svolení*“ (vlastní překlad).

V návaznosti na kritiku, kterou Battery API sklidilo, se Mozilla rozhodla odstranit podporu ve Firefox 52 (nedopatřením však odstranili pouze metodu `navigator.getBattery` a nikoliv rozhraní `BatteryManager`, což napravili ve verzi 72) [23].

1.2.2 Gamepad API

Gamepad API vzniklo v rámci HTML5 přidáním objektu `Gamepad` a rozšířením objektů `window` a `navigator`. Poskytuje vývojářům informace o připojených herních ovladačích a jejich stavu. Tyto informace zahrnují oznámení o připojení nebo odpojení ovladače, seznam dostupných tlačítek a informace

o jejich stisknutí, identifikační číslo produktu ovladače, jméno použitého driveru [24] a další.

Mozilla uvádí, že z důvodu fingerprintingu jsou ve Firefoxu použité gamepady viditelné pouze tehdy, pokud s nimi uživatel interaguje na dané webové stránce [25]. Podobný přístup zaujal i Google Chrome [26]. Toto opatření bylo zavedeno, přestože představuje nepříjemnost pro vývojáře [27]. Jedná se tak o jasnou a vítanou snahu zajistit uživatelům více soukromí. Gamepad API samozřejmě i přes toto opatření k fingerprintingu použít lze, ovšem bez větších problémů pouze na stránkách, které herní ovladače nějakým způsobem využívají i k legitimním účelům.

1.2.3 WebRTC API

„*WebRTC (Web Real-Time Communication – Webová komunikace v reálném čase) je technologie umožňující webovým aplikacím a stránkám zachytávat a streamovat audio a/nebo video media nebo vyměňovat jakákoliv arbitrární data přímo mezi dvěma prohlížeči, tj. bez potřeby prostředníka*“ (vlastní překlad) [28]. Umožňuje jednodušší a spolehlivější online konference a další P2P komunikaci. WebRTC je v současnosti podporován všemi obvyklými prohlížeči (Edge, Firefox, Chrome, Safari, Opera a vetšinou mobilních prohlížečů).

Jak ukazuje například nástroj Browerleaks [29], protokol WebRTC lze snadno zneužít pro fingerprinting. Poskytuje totiž vývojářům informace mimo jiné o soukromé IP adrese uživatele. Spojení dvou zařízení, u nichž je známá pouze veřejná IP adresa, by nebylo vždy možné, proto WebRTC využívá rozhraní ICE (Interactivity Connectivity Establishment), které prostřednictvím procesu zvaného *Candidate Gathering* zjistí mimo jiné i soukromou IP adresu a poskytne ji webové stránce. Toto je pochopitelně velký zásah do soukromí, ale také se jedná o účinný nástroj, pokud jde o fingerprinting. Proces odhalení soukromé IP adresy v demonstrovaném útoku popisuje podrobně bakalářská práce Bc. Jakuba Dvořáka [30].

Zajímavý přístup k řešení problému se soukromím v souvislosti s WebRTC zvolili vývojáři webového prohlížeče Safari. Safari umožní získaní soukromé IP adresy pouze v tom případě, že uživatel udělil souhlas s nahráváním videa a/nebo zvuku [31]. Tento přístup poskytuje uživateli určité množství očekávaného soukromí, na druhou stranu do jisté míry omezuje funkčnost WebRTC, neboť tento protokol by mělo být možné využívat i pro přenos jiných dat než audiovizuálních. Ze strany Safari se v tomto ohledu jedná o kompromis.

1.2.4 Touch API

JavaScriptové události TouchEvent umožňují vývojářům pracovat s informacemi o dotyku uživatele na dotykové obrazovce, případně informace o pohybu elektronického pera stylus [32]. Podporují i složitější gesta jako dotyk více prstů najednou, při kterém je API schopno informovat o začátku a konci do-

tyku každého jednotlivého prstu, stejně jako o změně pozice (tj. pohybu) prstů. Toto API lze v praxi využít například pro podporu kreslení na dotykové obrazovce prstem.

Specifikace [33] uvádí, jaká data jsou prostřednictvím tohoto API k dispozici; událost dotyku zprostředkovává mimo jiné informace o souřadnicích dotyku na obrazovce (`screenX` a `screenY`) a souřadnicích v rámci viditelné oblasti stránky se započítaným offsetem způsobeným scrollováním (`pageX` a `pageY`) nebo bez tohoto offsetu (`clientX` a `clientY`). K dispozici je také informace o přibližné ploše dotyku (`radiusX`, `radiusY` a `rotationAngle` jsou proměnné vymezující elipsu, která nejpřesněji ohraničuje plochu dotyku).

Tyto informace (především potom souřadnice dotytku na obrazovce) mohou – obzvláště na mobilních zařízeních, kde se uživatel dotýká plochy na stránkách neustále – prozradit skutečné rozměry obrazovky používaného zařízení [34]. Jak bylo zmíněno v předchozím textu, tato informace je samozřejmě využitelná pro vytvoření otisku.

1.2.5 Další JavaScriptová API

JavaScript poskytuje celou řadu dalších nástrojů, které lze pro fingerprinting zneužít. Mezi velmi rozšířené patří Canvas API a Web Audio API, které byly rozebrané v práci [7], a proto se jim nebudeme podrobně věnovat. Do výstupů, které tato API poskytují, se nepřímo promítají vlastnosti hardwaru uživatele a poskytují tak prostor pro relativně velmi unikátní otisk.

1.3 ETag a Last-Modified

Hlavičky `ETag` slouží ke zefektivnění práce s cache. Při zasílání zdroje může server pomocí hlavičky ETag připojit identifikátor verze (například hash) tohoto zdroje. Při dalším požadavku ze strany klienta se společně s dotazem pošle hlavička `If-None-Match` obsahující tento identifikátor. Pokud identifikátor souhlasí se stávající verzí, může server odpovědět statusem `304 Not Modified` (s tím, že vynechá tělo požadovaného zdroje a ušetří tak pásmo), podle kterého klient pozná, že verze uložená v cache je stále platná. Při změně zdroje musí server pochopitelně ETag aktualizovat (změnit na dosud nepoužitou hodnout). [35]

Formát identifikátoru posílaného v hlavičce `ETag` není nijak specifikován. Je na serveru, jaký (a jak dlouhý) identifikátor pro své zdroje zvolí. Rovněž může server rozhodnout, zda použije tzv. „slabý“ (weak) ETag, který odkazuje na verzi zdroje, ale nezaručuje identickou shodu (bajt po bajtu) dvou zdrojů se stejným ETagem [35]. Takový ETag je typicky snazší generovat, ale neslouží tak dobře pro potřeby cache.

Mechanizmus ETagu, je-li implementován na straně klienta, lze snadno zneužít pro sledování uživatele jako náhradu běžných cookies. Janc a Zalewski [36] tento proces vysvětlují; server pošle klientovi libovolnou hodnotu

a ten ji při další návštěvě automaticky posílá nazpět. Pokud server nevyužívá ETag jako identifikátor zdroje, ale jako identifikátor uživatele, jedná se v podstatě o ekvivalent chování běžné cookie.

Analogicky funguje starší, ale stále podporovaná obdoba hlavičky `ETag` – `Last-Modified`. Hlavní rozdíl je v tom, že hlavička `Last-Modified` udává datum a čas, kdy byla daná verze zveřejněna – tento časový údaj má předepsaný formát. Jak ale uvádí Janc a Zalewski [36], i při dodržení tohoto formátu je možné v rámci hlavičky `Last-Modified` uložit až 32 bitů informace. Zajímavostí také je, že většina současných prohlížečů předepsaný formát nekontroluje a bez ohledu na to, jakou hodnotu přijmou, při další návštěvě ji serveru pošlou zpět prostřednictvím hlavičky `If-Unmodified-Since`. V praxi se tedy tato hlavička dá využít v podstatě identicky jako ETag.

1.4 Kompresní algoritmus SDCH

Prostřednictvím hlavičky `Content-Encoding` [37] sděluje server klientovi, že obsah zaslaný v těle zprávy je komprimován a jaký kompresní algoritmus byl použit. Mezi nejčastější kompresní algoritmy patří v tomto kontextu Gzip a Brotli [38].

SDCH (Shared Dictionary Compression over HTTP) je kompresní algoritmus vyvinutý společností Google. Butler a spol. [39] uvádí, že běžně používané algoritmy sloužící ke kompresi HTTP odpovědí (například Gzip) využívají ke kompresi faktu, že se data opakují v rámci jedné odpovědi. Technika, která využívá ke kompresi data opakující se napříč větším množstvím odpovědí (typicky například hlavičku, zápatí a vložený JavaScriptový kód v HTML stránkách), se nazývá *delta kódováním* [39]. SDCH využívá pro implementaci delta kódování tzv. „sdílené slovníky“.

Sdílený slovník v SDCH je soubor, který server zašle uživateli – v jeho záznamech jsou uloženy textové řetězce, o kterých server předpokládá, že by se mohly často objevit v následujících HTTP odpovědích. Server potom v navazující komunikaci může v obsahu využívat reference do tohoto slovníku, které klient substituuje za odpovídající řetězec.

1.4.1 Využití SDCH jako supercookie a ukončení podpory

Janc a Zalewski [36] ukazuje, že SDCH lze využít jako nástroj pro vytvoření tzv. supercookie[1]. SDCH totiž využívá unikátní identifikátory pro rozpoznání sdíleného slovníku. Tyto identifikátory jsou stanoveny serverem a klient je při svých návštěvách posílá zpět serveru prostřednictvím hlavičky `Avail-Dictionary`. Toto chování je analogické s chováním běžných cookies a

[1]Supercookies jsou data sloužící k identifikaci uživatele, podobně jako běžné cookies. Rozdíl je v tom, že supercookies nevyužívají pro uchování identifikátoru úložiště určené pro cookies, ale zneužívají k tomu jinou funkcionalitu prohlížeče. Jedná se tak o termín velmi úzce související s fingerprintingem.

lze proto použít jako jejich náhrada. Zajímavé je, že tato podobnost s cookies je zmíněná už v návrhu SDCH [39], kde autoři uvádí: *„Slovníky jsou podobné s cookies, protože umožňují sdílet stav přes HTTP. Proto jsme modelovali slovníky po vzoru cookies. . .“* (vlastní překlad).

V Google Chrome data spojená s SDCH nezůstávala zachována po restartování prohlížeče – na druhou stranu ale zůstávala zachována mezi anonymním a běžným režimem prohlížeče, stejně jako mezi jednotlivými uživatelskými profily [36]. Google začal řešit tento problém spojený se soukromím roku 2013 [40] a roku 2017 SDCH odstranil z Google Chrome ve verzi 59 [41]. Jednou z uvedených motivací pro začátek diskuze o zrušení podpory SDCH byla právě obava spojená s narušením soukromí.

1.5 HPKP

HPKP (HTTP Public Key Pinning) je bezpečnostní mechanizmus, který prostřednictvím HTTP hlavičky informuje webového klienta o tom, že dotčenému webovému serveru náleží konkrétní kryptografický veřejný klíč [42].

1.5.1 X.509 certifikáty a význam HPKP

Za běžných okolností se při šifrovaných spojeních přes HTTPS (pomocí SSL nebo TLS při využití PKI – Public Key Infrastructure) na síti používá asymetrická kryptografie. Webový server využívá certifikátu, který je definován standardním formátem X.509 [43]. Typicky takový certifikát obsahuje kromě jiného:

- **Veřejný klíč** náležící serveru, který umožňuje klientovi poslat serveru zašifrovanou zprávu.
- **Digitální podpis** certifikátu, který vytvořila certifikační autorita.
- Podrobnější **informace o identitě serveru** a certifikační autoritě, která dotčený certifikát podepsala.

Systém PKI zajišťuje soukromí v komunikaci mezi serverem a klientem. Je k tomu však potřeba, aby klient důvěřoval certifikační autoritě, která podepsala certifikát webového serveru. Pokud je tato certifikační autorita kompromitována, může útočník začít vydávat falešné certifikáty a tak provést útok typu MITM (jako tomu bylo například v případu útoku na certifikační autoritu DigiNotar [44]). Spojení se serverem v tu chvíli přestávají být bezpečná. HPKP se snaží snížit riziko spojené s takovým útokem.

Pokud server využívá HPKP, pošle při první návštěvě klienta HTTP hlavičku s informací o tom, který veřejný klíč serveru náleží. Konkrétně jde o hash vytvořenou z SPKI – Subject Public Key Info [45], která je součástí některého certifikátu v certifikačním řetězci (a dále ještě jednu další – záložní –

hash jiného SPKI). Klient si tuto informaci uloží (na dobu, která je rovněž určena v hlavičce parametrem `max-age`). Při příští návštěvě klient očekává shodu mezi uloženým hashem a reálnými certifikáty, které obdrží. Pokud ke shodě nedojde, je upozorněn uživatel.

1.5.2 Využití HPKP jako supercookie

Systém HPKP lze zneužít pro sledování uživatelské aktivity. Tento proces vysvětlíme na následujícím scénáři:

1. Při prvním požadavku ze strany klienta K vytvoří server n-bitový řetězec S, který představuje (pokud možno unikátní) identifikátor klienta. Tento řetězec si uloží.

2. Server odešle klienta K na n různých dalších domén, které jsou ve vlastnictví provozovatele (například prostřednitvím linků na n obrázků, z nichž každý je uložen na jiné z cílových domén). Pro každou doménu existují dva odlišné validní certifikáty $C1$ a $C2$.

3. Pro i-tou doménu platí, že prostřednictvím HPKP klientovi K představí hash odpovídající certifikátu $C1$, pokud $S[i] = 0$. Naopak představí hash odpovídající certifikátu $C2$, pokud $S[i] = 1$.

4. Při druhém požadavku klienta K bude tento analogickým způsobem odeslán na n domén, stejných jako v bodě 2. Cílem serveru je nyní zjistit klientův identifikátor S, aby byl schopen přiřadit předchozí aktivitu tohoto uživatele k aktivitě nastávající.

5. Každá z dotčených domén se prokáže certifikátem $C1$. Pokud požadavek na i-tou doménu selhal, znamená to, že se tato doména posledně prokázala hashem náležejícím k certifikátu $C2$, a tudíž $S[i] = 1$. Pokud požadavek neselhal, platí opačný případ a $S[i] = 0$. Tímto způsobem jsme schopni zrekonstruovat všech n bitů řetězce a získat zpět platný identifikátor.

Je zřejmé, že tímto způsobem lze vytvořit velmi silný identifikátor s (teoreticky) neomezenou velikostí a lze tak (opět teoreticky) nahradit standardní cookie. Technicky je možné scénář popsaný výše realizovat jako n subdomén jedné domény s využitím dvou takzvaných wildcard certifikátů (certifikáty, které mohou být uplatněny na více než jednu subdoménu).

1.5.3 Problémy HPKP a ukončení podpory

Přestože HPKP bylo navrženo jako rozšíření, které napomáhá síťové bezpečnosti, ukázalo se, že v praxi funguje spíše kontraproduktivně. Prohlížeče postupně začaly upouštět od podpory HPKP – důvodem byla nízká popularita

(jak uvádí Helme [46], koncem roku 2019 využívalo HPKP pouze 650 stránek ze seznamu Alexa Top 1 Million[2]), ale především také obtížná implementace ze strany provozovatelů webových služeb a rizika, která HPKP přinesla – patřily mezi ně tzv. HPKP Suicide a RansomPKP.

1.5.3.1 HPKP Suicide

HPKP Suicide (sebevražda skrz HPKP) je termín, jenž označuje situaci, při které vývojáři nastaví svou HPKP hlavičku na nějakou hodnotu veřejného klíče, která je buď neplatná, nebo patří ke klíči, k němuž ztratili přístup [48]. Pokud k tomu dojde a uživatelé mezitím stihnou navštívit provozovaný server, stanou se stránky pro tyto uživatele nepřístupnými až do doby vypršení parametru `max-age`. Provozovatel stránek pak nemá prostředky, jak problém napravit, a může jen čekat.

1.5.3.2 RansomPKP

RansomPKP označuje situaci, při níž útočník získá vládu nad nějakým serverem a využije HPKP Suicide proti jeho provozovateli. Útočník na kompromitovaném serveru nastaví HPKP hlavičku tak, aby ukazovala na útočníkovy klíče. Helme [49] uvádí následující kód jako ukázku škodlivého HPKP:

```
1 Public-Key-Pins: max-age=31536000; includeSubDomains;
2 pin-sha256= LOCKOUT_KEY;
3 pin-sha256= RANSOM_KEY;
```

`LOCKOUT_KEY` odkazuje na klíč, který bude podepsaný nějakou certifikační autoritou, je generovaný na serveru a útočník ho bude často měnit, dokud vlastník stránky nezíská zpět kontrolu. `RANSOM_KEY` naopak odkazuje na klíč, který útočník uchová v tajnosti a je po celou dobu útoku neměnný. V moment, kdy provozovatel získá server zpět pod svou kontrolu, má potenciálně velké množství uživatelů nastavené hodnoty HPKP na zfalšovanou hodnotu (a tudíž mají v podstatě omezený přístup k webové službě) a klíče k většině z nich nemůže provozovatel získat zpátky (všechny až na poslední verzi `LOCKOUT_KEY` jsou ztraceny). Útočník se pak může poškozenému serveru pokusit prodat `RANSOM_KEY`, který kvůli skutečnosti, že celou dobu zůstal nezměněn, dokáže vyřešit pro provozovatele celý problém se ztrátou uživatelů. Ale i pokud provozovatel klíč koupí, útočník má k němu stále přístup, tudíž některé bezpečností problémy přetrvávají [48]. Závažnost útoku může útočník zvýšit pomocí parametrů HPKP:

– Nastavením `includeSubDomains` se útok vztáhne nejen na jednu postiženou doménu, ale také na všechny její subdomény.

[2] Amazon prostřednictvím svého API zveřejňuje seznam webových stránek, které jsou podle jejich měřítek nejpopulárnější [47].

– Díky parametru `max-age` má útočník pod kontrolou, jak dlouho bude trvat, než uložená hodnota HPKP v prohlížečích uživatelů expiruje a běžný chod stránek se obnoví.

Kvůli výše popsaným problémům Google Chrome verzí 72 odstranil podporu HPKP [50]. Mozilla Firefox ve verzi 72 přistoupila ke stejnému kroku [51] (s tím, že podporu lze v prohlížeči znovu obnovit v nastavení zapnutím preference `security.cert_pinning.hpkp.enabled`, která je ve výchozím stavu vypnutá [52]). Uživatelé tak získali i větší anonymitu, pokud jde o sledování uživatelské aktivity pomocí fingerprintingu, byť v tomto případě spíše jaksi mimochodem.

1.6 HSTS

HSTS (HTTP Strict Transport Security) je „*hlavička v HTTP odpovědích, která umožňuje webovým stránkám sdělit prohlížeči, že by měly být navštěvovány pouze prostřednictvím HTTPS namísto HTTP*“ (vlastní překlad) [53].

Uvažujme webovou stránku, která podporuje šifrované spojení. Při návštěvě této stránky prostřednictvím nešifrovaného kanálu (HTTP) lze spojení přesměrovat (typicky pomocí 301 Moved Permanently) na HTTPS. Znamená to ale, že pokud klient přistupuje ke stránce pokaždé nejprve přes HTTP (což se v praxi děje často, neboť uživatelé většinou při psaní URL vynechávají prefix `https://` a výchozí protokol je ve většině prohlížečů HTTP), první část komunikace (až do navázání šifrovaného spojení) vždy probíhá přes nezabezpečený kanál. Tohoto faktu využívá nástroj SSLStrip výzkumníka Moxieho Marlinspika [54].

SSLStrip lze využít k útoku typu MITM, při kterém se uživatel domnívá, že server nepodporuje šifrované spojení (přestože to není pravda), a útočník může číst a dokonce měnit veškerou komunikaci probíhající mezi obětí a serverem. HTTP Strict Transport Security bylo navrženo jako bezpečnostní opatření, které výrazně redukuje nebezpečí spojené s tímto typek útoku.

Prostřednictvím hlavičky HSTS sdělí server (podobně jako v případu HPKP typicky při první návštěvě) klientovi, že podporuje šifrovanou komunikaci a žádá tím, aby prohlížeč tuto informaci uložil a využil jí při dalších návštěvách. Klient tak bude při další návštěvě serveru přistupovat přímo přes HTTPS a SSL stripping pak už není možný. Hlavička HSTS, podobně jako hlavička HPKP, obsahuje direktivu `max-age`, která určuje, jak dlouho má prohlížeč uchovávat informaci o tom, že server podporuje šifrované spojení. Dále také obsahuje nepovinnou direktivu `includeSubDomains`, která klienta informuje o tom, že se hlavička vztahuje rovněž na všechny subdomény.

1.6.1 Využití HSTS jako supercookie

Proces zneužití Strict Transport Security pro sledování uživatelské aktivity je analogický jako v případě HPKP:

1. Při prvním požadavku ze strany klienta K vytvoří server n-bitový řetězec S, který představuje identifikátor klienta.
2. Server odešle klienta K na n různých dalších domén, které jsou ve vlastnictví provozovatele.
3. Pro i-tou doménu platí, že v odpovědi pošle platnou HSTS hlavičku, pokud $S[i] = 1$. Naopak nepošle tuto hlavičku, pokud $S[i] = 0$.
4. Při druhém požadavku klienta K bude tento analogickým způsobem odeslán na n domén, stejných, jako v bodě 2. Cílem serveru je nyní zjistit klientův identifikátor S.
5. Pokud klient K přistupuje na i-tou doménu prostřednictvím HTTPS, znamená to, že v minulosti od této domény přijal HSTS hlavičku, a tudíž $S[i] = 1$. V opačném případě $S[i] = 0$. Tímto způsobem zrekonstruujeme celý identifikátor S.

Zajímavostí je, že autoři HSTS předvídali tuto techniku sledování uživatelů a vysvětlili ji už v RFC 6797 [55], který standard HSTS popisuje. Uvádějí tam: „*Ti, kteří kontrolují jeden nebo více HSTS serverů, mohou zakódovat informaci do doménových jmen, jež vlastní, a způsobit tak, že si klient tuto informaci uloží do cache (. . .) Taková technika může být potenciálně zneužita jako další forma webového sledování*“ (vlastní překlad).

Otisk vytvořený pomocí HSTS bylo dokonce možné využít například v Google Chrome i napříč běžným a anonymním režimem prohlížeče. Informace o tom, které servery využívají HTTPS, se totiž v Chrome přenášela ze standardního do anonymního režimu (ale ne naopak) [36]. Toto chování bylo odstraněno v Google Chrome s verzí 64 [56].

Přestože je známou skutečností, že HSTS lze využít k fingerprintingu, jedná se o natolik hodnotný mechanizmus z hlediska bezpečnosti, že jej v současnosti podporují všechny běžné prohlížeče (Internet Explorer, Edge, Firefox, Chrome, Safari, Opera a drtivá většina mobilních prohlížečů [57]) a statistika [58] udává, že hlavičku HSTS používá zhruba 12,6 % webových stránek, které se umístily v Alexa Top 10 million.

1.7 Fingerprinting pomocí kaskádových stylů

JavaScript nabízí vývojářům velkou škálu možností, jak získat otisk uživatele. Moderní prohlížeče však stále nabízí svým uživatelům možnost JavaScript vypnout, čímž se fingerprint pochopitelně významně změní (i když anonymizaci

to posloužit nemusí – uživatelů s vypnutým JavaScriptem bude totiž typicky velmi málo a rozlišit mezi nimi je stále možné fingerprintingovými technikami, které JS nevyžadují).

Jak uvádí Takei a spol. [59], podstatná část informací, které mohou stránky využívat k získaní otisku prohlížeče pomocí JavaScriptu, lze shromáždit i za použití CSS. V této podkapitole představíme, jakým způsobem toho lze docílit.

1.7.1 Fingerprinting seznamu fontů

Seznam fontů, které jsou na systému nainstalované, může být (obzvláště pro desktopové uživatele) velmi silným identifikátorem. Práce [6] představuje ve své práci čtyři různé způsoby, jak získat seznam fontů, které jsou nainstalované na systému uživatele. Jedním z nich je detekce využívající kombinaci kaskádových stylů a JavaScriptu, kterou implementuje například nástroj [60]. Tento mechanizmus probíhá následovně:

1. Vytvoříme blokové elementy $E1$ a $E2$, které obsahují stejné neprázdné řetězce znaků.
2. S využitím CSS vlastnosti `font-family` nastavíme písmo elementu $E1$ na některý z běžných fontů, v našem příkladu zvolíme `Arial`.
3. Pomocí JavaScriptu změříme šířku elementu $E1$.
4. S využitím `font-family` nastavíme písmo elementu $E2$ na seznam dvou fontů – prvním z nich bude některý z méně běžných fontů (tj. font, jehož přítomnost na systému uživatele chceme detekovat), druhým fontem bude opět `Arial`.
5. Pomocí JavaScriptu změříme šířku elementu $E2$.
6. Porovnáme naměřené hodnoty šířek elementů $E1$ a $E2$. Pokud se liší, zkoumaný font je nainstalován.

Tato metoda využívá faktu, že CSS vlastnost `font-family` umožňuje předat hodnotou seznam fontů (namísto jednoho) [61]. Pokud první font na seznamu není nainstalovaný, použije se pro vykreslení druhý (pokud není ani ten, využije se třetí atp.).

Výhodou využití kombinace CSS a JavaScriptu je skutečnost, že celý otisk prohlížeče je sestaven na straně klienta, bez nadbytečné záteže na serveru, kterému se výsledek potom pouze oznámí. Na druhou stranu tento princip nelze využít, pokud klient nepovoluje spouštění JavaScriptu. Jak si ale ukážeme na následujícím příkladu, přítomnost JavaScriptu není pro detekci fontů nezbytně nutná. Uvažujme výpis kódu 1.1, který je inspirován zdrojovým kódem [62] demonstrujícím tuto techniku. V tomto příkladu usilujeme o zjištění, zda má uživatel nainstalovaný font „Comic Sans“.

Na řádcích 1–4 definujeme pomocí `@font-face` [63] nový font s názvem `myFont`. Pomocí funkce `url()` deklarujeme, že chceme, aby se tento font stáhl z adresy, kterou předáváme jako argument. Tato adresa ovšem směřuje na stránku, která do databáze uloží záznam o tom, že uživatel nemá nainstalovaný font Comic Sans.

```
1 @font-face {
2   font-family : myFont;
3   src : url("/fingerprinting/fonts?font=ComicSans");
4 }
5
6 #detection_element {
7   font-family : "Comic Sans", myFont;
8 }
```

Výpis kódu 1.1: Detekce přítomnosti fontu pomocí CSS

Když se uživateli vykresluje element `#detection_element`, pokusí se prohlížeč najít Comic Sans mezi nainstalovanými fonty. Pokud se mu to nepodaří, využije fontu, který je na seznamu `font-family` uvedený na druhém místě – tím je ovšem námi definovaný font `myFont`. Prohlížeč v tuto chvíli načítá url `/fingerprinting/fonts?font=ComicSans` a odesílá tak na server informaci o nepřítomnosti fontu Comic Sans.

Samotné pravidlo `@font-face` umožňuje definovat více zdrojů jednoho fontu – pomocí funkce `local()` se aplikuje font specifikovaného jména, pokud je nainstalovaný na zařízení uživatele, pomocí funkce `url()` pak lze určit adresu, na které je font ke stažení v případě, na zařízení nainstalovaný není. Zřetězením těchto dvou funkcí lze analogickým způsobem detekovat, zda je font nainstalovaný.

Pokud tuto techniku použijeme na větší množství fontů, můžeme získat velmi ojedinělý otisk napříč uživateli. O každém fontu, který se nepodaří načíst, nás informuje prohlížeč uživatele, u všech ostatních testovaných fontech můžeme předpokládat, že se je načíst podařilo, a tudíž jsou na systému nainstalované.

1.7.2 Detekce rozlišení obrazovky uživatele

Podobně jako detekce seznamu fontů, zjištění rozlišení prohlížeče je mnohdy používáno jako fingerprintingová strategie – využívají ho například i projekty Panopticlick [64] a AmIUnique [65]. JavaScript umožňuje vývojářům tuto informaci získat pomocí členských proměnných `window.screen.height` a `window.screen.width`, které mají uložené přímo hodnoty rozlišení.

Kaskádové styly rovněž umožňují detekovat rozlišení [59], byť komplikovanějším a méně elegantním způsobem. Jednoduchý příklad můžeme vidět na výpisu kódu 1.2, který je inspirován zdrojovým kódem [62] demonstrujícím tuto techniku.

```
@media (min-width: 1024px) {
  #element::after {
    content: url("/fingerprinting/width?width=1024px");
  }
}

@media (min-width: 1280px) {
  #element::after {
    content: url("/fingerprinting/width?width=1280px");
  }
}
```

Výpis kódu 1.2: Detekce rozlišení obrazovky pomocí CSS

Tato technika využívá tzv. *Media features* [66], konkrétně `min-width` a `min-height`. Tyto ovšem korespondují s rozměry okna prohlížeče a dávají proto v tomto kontextu dobré výsledky pouze tehdy, je-li prohlížeč v maximalizovaném režimu. Pokud tomu tak není, lze s výhodou využít `min-device-width` a `min-device-height`, které přímo korespondují s rozlišením obrazovky. Je ovšem potřeba dodat, že jsou považovány za zastaralé [66].

Pokud server využívající techniku demonstrovanou na výpisu kódu 1.2 přijme dotaz na adrese `/fingerprinting/width?width=1024px`, lze předpokládat, že šířka obrazovky sledovaného uživatele je mezi 1024 a 1279 pixely. Analogickým způsobem lze rozšířit seznam detekovaných rozměrů a zpřesnit tak vytvořený otisk – granularita už závisí pouze na vývojáři a lze zjemnit teoreticky až na jeden pixel. Stejným způsobem, jakým lze detekovat šířku zařízení, je možné pochopitelně detekovat i jeho výšku.

1.7.3 Detekce orientace okna prohlížeče

Podobně jako výšku a šířku lze pomocí kaskádových stylů detekovat i *orientaci* okna prohlížeče [66] – tj. zda je prohlížeč orientovaný na výšku nebo na šířku (režim portrét nebo režim krajina). Výpis kódu 1.2 ukazuje způsob, jak takový otisk obstarat.

```
@media (orientation: landscape) {
  #element::after {
    content: url("/fingerprinting/orient?o=landscape");
  }
}
```

Výpis kódu 1.3: Detekce orientace obrazovky pomocí CSS

Ve většině případů tento otisk nepřinese velkou hodnotu, protože většina uživatelů mobilních zařízení bude vykazovat hodnotu obrazovky otočené na výšku a většina desktopových uživatelů naopak na šířku. Význam této techniky však vyvstává v následujících situacích:

– Uživatel používá desktopové zařízení a obrazovka je orientovaná na výšku, nebo používá mobilní zařízení a obrazovka je orientovaná na šířku.

– Otisk orientace obrazovky neodpovídá otisku rozlišení obrazovky – tento případ mohl být například způsoben nedokonalým pokusem o obranu proti fingerprintingu.

1.7.4 Detekce implementace CSS vlastností

Otisk použitého prohlížeče lze udělat na základě seznamu CSS vlastností, které jsou v dané verzi implementovány. Takei a spol. [59] uvádí, že vykreslovací engine typicky ignoruje výrazy CSS, které nedokáže interpretovat (protože obsahuje CSS vlastnosti, které nezná).

Různé prohlížeče využívají pro nestandardní nebo experimentální vlastnosti CSS vlastní prefixy. Takové vlastnosti mohou dobře posloužit pro potřeby fingerprintingu. Mezi nejčastější prefixy patří [67] [68]:

- **-ms-** – Prohlížeče společnosti Microsoft (Internet Explorer a Microsoft Edge)
- **-moz-** – Mozilla Firefox
- **-o-** – Starší verze Opery
- **-webkit-** – Prohlížeče založené na WebKit – Chrome, Safari a novější verze Opery

Ve výpisu kódu 1.4 můžeme vidět ukázku testování přítomnosti CSS vlastnosti `linear-gradient` vybranou z demonstračního kódu [59]:

```
#elem {
  background : linear-gradient(to right,
    rgba(255,255,255,0),
    rgba(255,255,255,1)
  ),
  url("/fingerprinting/css_features?f=gradient");
}
```

Výpis kódu 1.4: Detekce přítomnosti funkce linear-gradient() v CSS

Pokud prohlížeč interpretující kód 1.4 nezná funkci `linear-gradient`, bude se jako pozadí elementu `#elem` interpretovat zdroj umístěný na URL `/fingerprinting/css_features?f=gradient`, na které pochopitelně server eviduje otisk uživatele.

Pomocí CSS pravidla `@supports` [69] lze stejného výsledku docílit i systematičtějším způsobem. Toto pravidlo umožňuje specifikovat deklarace CSS na základě toho, jestli dotčený prohlížeč podporuje vlastnost, kterou `@supports` přijímá jako argument. Na druhou stranu samotný `@supports` není podporovaný ve všech prohlížečích (například v žádné verzi prohlížeče Internet Explorer [69]), což může v případě krajních situací entropii otisku snížit.

1.7.5 Měření doby před stisknutím odkazu

Způsob, jakým se uživatel chová na webových stránkách, lze evidovat a teoreticky podrobit fingerprintingu [34]. Jako příklad uvádí Perry a spol. [34] frekvenci stisknutí tlačítek při psaní textu, pohyby myší a rychlost při klikání na interaktivní prvky stránek. Poslední jmenovaný lze sledovat pomocí kaskádových stylů [62] a pravidla `@keyframes` [70]. Na výpisu kódu 1.5, který je upraveným kódem od Böhmera [62], můžeme vidět fungování této fingerprintingové techniky.

Když uživatel najede kurzorem myši na element `#elem`, aktivuje se animace definovaná pomocí `@keyframes` na prvním až šestém řádku. Délka animace je v tomto případě stanovená na pět sekund – tento čas ohraničuje nejdelší možný sledovaný úsek. Animace načítá URL odkazující na stránku evidující otisk uživatele po 33, 66 a 100 procentech ze stanovené délky animace. Opět je možné zmenšit granularitu (a prodloužit sledovaný úsek) a tím získat přesnější měření.

```
1  @keyframes pulsate {
2      0% { background-image : url("/fingerprint?t=0") }
3      33% { background-image : url("/fingerprint?t=33") }
4      66% { background-image : url("/fingerprint?t=66") }
5      100% { background-image : url("/fingerprint?t=100") }
6  }
7
8  #elem:hover::after {
9    animation: pulsate 5s infinite;
10   animation-name: pulsate;
11   animation-duration: 5s;
12   content: url("/fingerprint/anim?t=NULL");
13 }
```

Výpis kódu 1.5: Detekce rychlosti kliknutí myši pomocí CSS

1.8 Fingerprinting pomocí HTTP autentizace

Protokol HTTP poskytuje vývojářům mechanizmus pro přihlašování uživatelů. Běžný scénář přihlášení uživatele na webové stránce pomocí základního schématu („Basic Auth“) vypadá následovně [71]:

1. Klient se pokusí přistoupit ke zdroji, který je dostupný pouze pro přihlášené uživatele.

2. Server odpoví statusem `401 (Unauthorized)` a poskytne v odpovědi informaci o tom, jakým způsobem se má uživatel autentizovat. K tomu využívá hlavičku `WWW-Authenticate`, která obsahuje informaci o typu přihlášení a popis udávající, k užívání čeho se uživatel přihlašuje.

3. Klient, chce-li provést autentizaci, zašle na server požadavek s hlavičkou `Authorization`, která respektuje typ přihlášení udaný serverem a obsahuje přihlašovací údaje (přihlašovací údaje typicky zadává uživatel do dialového okna vytvořeného prohlížečem).

4. Pokud přihlášení proběhne úspěšně a uživatel je autorizován k přístupu ke zdroji, server odpoví statusem `200 (OK)` – v opačném případě odpoví `403 (Forbidden)`.

Prohlížeče mají typicky vlastní dialogové okno připravené pro potřeby autentizace uživatele přes HTTP. James [72] však představuje způsob, jakým je možné využít JavaScript pro přihlášení uživatele – cílem je dát vývojářům možnost upravit vzhled přihlašovacího okna pomocí HTML a CSS s tím, že funkcionalita vytvoření a odeslání požadavku pro přihlášení je delegována JavaScriptu.

Grossman [73] uvádí, jakým způsobem je možné využít HTTP autentizace prostřednictvím JavaScriptu pro potřeby krátkodobé identifikace:

1. Při první návštěvě pošle server klientovi webovou stránku se standardním obsahem a připojeným JavaScriptem, který klienta odešle k HTTP autentizaci. Jméno uživatele a heslo je však předem vybrané serverem a uživatel se vůbec nedozví, že k autentizaci došlo. Dvojice jméno a heslo představují identifikátor uživatele, který server bude nadále používat pro sledování jeho aktivity.

2. Poté, co je klient bez vědomí uživatele autentizován, uloží si prohlížeč přihlašovací údaje do cache.

3. Prohlížeč bude připojovat přihlašovací údaje (které jsou však zároveň identifikátorem klienta) ke každému následujícímu požadavku směrem k serveru.

1.9 Fingerprinting chování a preferencí uživatele

Přestože fingerprinting chování uživatele není doslova vzato součástí oblasti fingerprintingu prohlížeče, jedná se o souhrn technik, které prohlížeč v jistém smyslu umožňuje (především prostřednictvím JavaScriptu), proto je pro úplnost v této práci rozebereme také. Jedná se navíc o techniky, které (byť nemusí být vždy zcela spolehlivé) umožňují v některých případech sledování uživatele dokonce napříč vícero zařízeními.

1.9.1 Interakce uživatele s myší a klávesnicí

Až na výjimky každý desktopový uživatel na internetu používá myš (nebo touchpad) i klávesnici, a to velmi frekventovaně. Ukazuje se, že způsob, jakým

uživatel pracuje s těmito vstupními periferiemi, je do značné míry jedinečný pro každého člověka a s odpovídajícími nástroji tedy lze využít k jeho identifikaci.

Zach Jorgensen a Ting Yu ve své studii [74] provedli několik experimentů, v nichž se snažili určit identifikovatelnost člověka na základě jeho způsobu práce s myší a touchpadem (pro potřeby autentizace). Ukázalo se, že mezi uživateli existují detekovatelné rozdíly v chování – byť k tomu, aby tyto rozdíly bylo možné spolehlivě rozpoznat, je potřeba shromáždit od zkoumaných subjektů podstatné množství dat.

Je ovšem důležité poznamenat, že cílem autorů bylo pomocí interakce s myší identifikovat konkétní osoby, bez ohledu na takzvané „proměnné prostředí“ (environmental variables). Těmi mohou být například model myši, který uživatel použil, rozlišení obrazovky nebo rychlost myši nastavitelná v operačním systému. Vliv těchto proměnných snižuje možnost identifikovat konkrétní osoby nezávisle na prostředí.

Proměnné prostředí ale naopak zvyšují šanci uživatele rozpoznat, pokud se lze spolehnout na to, že pro každého uživatele je toto prostředí neměnné (demonstrace podstatného vlivu prostředí na výsledky měření je jedním z hlavních závěrů práce [74]). Jelikož lze předpokládat, že běžný návštěvník stránek bude napříč jednotlivými návštěvami pracovat ve velmi podobném prostředí (bude používat stejnou myš apod.), můžeme dovodit, že prostor pro fingerprinting pomocí této techniky je značný.

Podobný experiment provedli Ingo Deutschmann a spol. [75], kteří zkoumali možnosti autentizace uživatelů počítače mimo jiné na základě používání myši a také dynamiky stisku tlačítek na klávesnici. Závěrem tohoto experimentu uvádí, že obě metriky jsou důvěryhodným mechanizmem pro autentizaci osob.

1.9.2 Pohyb a rotace mobilního zařízení

Uživatelé mobilních zařízení obvykle nepoužívají myš a tak podstatná část sekce 1.9.1 není pro jejich fingerprinting využitelná. Naproti tomu existuje JavaScriptové API, které umožňuje velkou část mobilních zařízení identifikovat pomocí jejich orientace a pohybu v prostoru.

Naprostá většina chytrých telefonů v současnosti obsahuje akcelerometr. Jak uvádí Sharma [76], jedním z jeho běžných využití je rozpoznání rotace mobilního zařízení pro potřeby otočení obrazu (změna režimu obrazu na výšku a na šířku). JavaScriptem lze získat data z akcelerometru pomocí událostí `deviceorientation` a `devicemotion` [77].

Přímočaré využití tohoto JavaScriptového API pro fingerprinting je například v kombinaci s informací o orientaci webové stránky. Jak jsme ukázali v sekci 1.7.3, orientaci okna prohlížeče lze detekovat například pomocí kaskádových stylů. Pokud tuto informaci spojíme s reálnou orientací mobilního zařízení v prostoru, můžeme například usuzovat na skutečnost, zda má uživatel

blokované *otočení obrazu*. Například pokud akcelerometr udává, že je zařízení otočené o devadesát stupňů (tedy na šířku), ale orientace obrazovky je stále na výšku, pravděpodobně je otočení obrazu zakázané (tato situace může reálně nastat například pokud uživatel mobil používá vleže). Detekce této situace může pomoct udělat relativně unikátní otisk, byť často zřejmě jen dočasný.

Data z akcelerometru však teoreticky lze využít pro daleko rozsáhlejší fingerprinting. Aviv a spol. [78] udávají, že data z akcelerometru vytváří velkou plochu pro únik informací postranním kanálem, neboť z drobných změn v orientaci zařízení lze s určitou přesností odvodit, na jakém místě se uživatel dotkl obrazovky zařízení nebo stiskl tlačítko.

Aviv a spol. [78] se ve svém experimentu snaží odhadovat vstupní PIN do mobilního zařízení pouze s využitím dat z akcelerometru při zadávání tohoto kódu. Na základě tohoto experimentu odhadují, že s jejich technikou predikce pomocí strojového učení lze očekávat až 277krát větší šanci na uhodnutí správného kódu PIN na první pokus než při náhodném hádání (tedy zhruba 2% pravděpodobnost při odhadování správného kódu mezi 10 000 čísly, což odpovídá počtu všech možností běžných čtyřmístných kódů PIN). Dále odhadují, že pro zámky chráněné vstupním vzorem je pravděpodobnost správného odhadu na první pokus při použití jejich metody dokonce 23597krát větší než při náhodném hádání (tj. zhruba 6 %).

Tato skutečnost, kterou potvrzují i další články [79] [80], má samozřejmě dopad na větší množství oblastí počítačové bezpečnosti a fingerprinting uživatelů mobilních zařízení je jednou z nich.

1.9.3 Tlak uživatele na dotykovou obrazovku

V sekci 1.2.4 jsme si představili události TouchEvent, prostřednictvím nichž lze v JavaScriptu shromažďovat informace o dotyku uživatele na dotykových obrazovkách, což může pomoci vytvořit si představu o rozlišení obrazovky zařízení. Toto JavaScriptové API ale poskytuje kromě souřadnic a plochy dotyku také data o tom, jak velkou silou uživatel tlačil prstem na obrazovku. Tato informace je udávána proměnnou `force`, která může nabávat hodnot od 0 do 1, přičemž 0 reprezentuje „žádný dotyk“ a 1 reprezentuje nejvyšší možný tlak, který je dotykové zařízení schopno rozpoznat [33].

Některé studie [81] [82] ukazují, že data o tlaku vyvinutém při používání zařízení je možné použít jako biometrický údaj, který zpřesňuje autentizaci uživatele například na základě frekvence úhozů při psaní. Je tedy nasnadě, že lze tyto informace teoreticky v nějaké míře využít i při fingerprintingu. Je ovšem potřeba dodat, že různá zařízení mohou udávat různou informaci o tlaku, přestože skutečný tlak je stejný, a to kvůli vnitřnímu nastavení a možnostem senzorů zařízení. To může být příčinou dostatečně velkého šumu, aby uživatel například nebyl identifikovatelný napříč vícero zařízeními, což zneužitelnost tohoto API snižuje.

1.9.4 Zásahy do vzhledu a výchozího chování stránek

Uživatelé mohou dobrovolně pozměnit výchozí chování prohlížeče nebo upravit vzhled navštívěných stránek, aby používání internetu více odpovídalo jejich preferencím. Tyto úpravy mohou být v některých případech detekovatelné, typicky opět JavaScriptem. Janc a Zalewski [36] uvádí několik způsobů, jak uživatelé mohou upravit chování prohlížeče tak, aby bylo možné tyto úpravy detekovat a zneužít k fingerprintingu:

- Změna vzhledu stránek kvůli dostupnosti, což zahrnuje změnu výchozího přiblížení stránek, změnu nastavení barvy a velikosti textu nebo jakéhokoliv zásahu do kaskádových stylů stránek.
- Blokování vyskakovacích oken, blokování některých částí obsahu (například bannerů a reklam).
- Změna vlastností prohlížeče jako například nastavení hlavičky DNT (hlavička Do Not Track může být zasílaná s požadavky na server a sděluje serveru, že si uživatel nepřeje, aby byla jeho aktivita sledována [83]).

1.10 Časová analýza postranního kanálu

Časový útok v počítačové bezpečnosti obecně umožňuje útočníkovi zjistit nějakou tajnou informaci pouze na základě množství času, který byl potřeba pro provedení vybrané operace zkoumaného zařízení. Časovou analýzu lze využít i ve fingerprintingu, a to hned na několika místech.

1.10.1 Cache uložená v prohlížeči

Pomocí časové analýzy je možné rozpoznat, zda má klient uložený nějaký zdroj v cache. Při fingerprintingu lze cílit na často navštěvované zdroje na internetu, které mají potenciál být oblíbené mezi návštěvníky, a odeslat na jejich URL požadavek [36]. Podle času, který klientovi zabralo zdroj načíst, lze odhadnout, zda byl uložen v lokální cache, nebo zda musel klient zdroj stáhnout. Fingerprinting tak zneužívá právě té vlastnosti cache, kvůli které byla navržena – a tou je úspora času. Otisk v tomto případě tedy vytváříme na základě seznamu často navštěvovaných zdrojů, které byly (podle výsledků časové anaýzy) uloženy v cache uživatele.

JavaScript poskytuje velmi přesné výsledky při měření času, který byl zapotřebí pro provedení nějaké akce. Mechanizmus CORS (Cross-Origin Resource Sharing) [84], který je implementován na současných verzích všech běžných prohlížečů, ovšem znemožňuje skriptům získávat jakékoliv informace o datech načítaných ze zdrojů, které jsou hostované na jiné doméně. Na druhou stranu je JavaScriptu umožněno (například prostřednictvím události `onload`) spouštět skripty ve chvíli, kdy se prohlížeči podaří načíst vložený (embedded)

obsah. Bortz a spol. [85] představují a porovnávají dva přístupy, jak změřit dobu potřebnou pro načtení externího zdroje:

- Prostřednictvím HTML tagu `IFRAME` lze načíst jinou HTML stránku. Pomocí události `onload` spustíme JavaScriptový kód, který změří aktuální čas ve chvíli, kdy se podaří načíst celý vložený obsah. Problémem tohoto přístupu je fakt, že typická webová stránka v rámci svého těla načítá další zdroje (kaskádové styly, JavaScript, obrázky a další). Tato skutečnost přidává do procesu měření velké množství nechtěného šumu, který přispívá k nepoužitelnosti této techniky.

- HTML umožňuje načíst jiný zdroj také prostřednictvím tagu `IMG`. Tento tag narozdíl od `IFRAME` načítá jen jeden zdroj, a to ten, na který ukazuje atribut `src`. `IMG` je ovšem navržen pouze pro načítání obrázků, nikoliv HTML stránek ani dalších zdrojů. Prohlížeč ale nemůže dopředu vědět, zda načítaný zdroj je skutečně obrázkem, a proto odešle standardní požadavek na vzdálený server. Teprve při přijetí odpovědi prohlížeč rozhodne, zda se jedná o obrázek, nebo ne – na případ, že načtený zdroj obrázkem není, lze reagovat spuštěním skriptu prostřednictvím události `onerror`. Kód zodpovědný za měření času lze tedy vložit do funkce vyvolané při chybě načítání obrázku a lze tak změřit čas potřebný pro načtení libovolného zdroje. Praktickou ukázku této techniky si lze prohlédnout na výpisu kódu 1.6.

```
<html><body><img id="test" style="display: none">
  <script>
    var test = document.getElementById('test');
    var start = new Date();
    test.onerror = function()
    {
      var end = new Date();
      var finalTime = end - start;
      alert("Cas pro nacteni zdroje: " + finalTime);
    }
    test.src = "http://www.example.com/page.html";
  </script>
</body></html>
```

Výpis kódu 1.6: Detekce doby načtení externího zdroje pomocí JavaScriptu. Převzato z [85] a upraveno.

Felten a Chneider [86] ukazují, že měření času, který byl zapotřebí pro vykonání požadavku, lze provést i na prohlížeči s vypnutým JavaScriptem, byť za dosažení nižší přesnosti. Tato technika spočívá v provedení tří požadavků:

1. Požadavek na stránku útočníka, který zaznamená aktuální čas.

2. Požadavek na zdroj, o kterém chceme rozhodnout, zda je uložen v cache prohlížeče.

3. Druhý požadavek na stránku útočníka, který zaznamená aktuální čas.

Odečtením času naměřeném v prvním a třetím kroku lze odhadnout doba, která byla zapotřebí k provedení požadavku v druhém kroku (za předpokladu, že provedení všech požadavků proběhlo sekvenčně).

1.10.2 Cross-site timing

Časová analýza doby nutné pro načtení zdroje popsaná v sekci 1.10.1 lze použít na obdobný typ fingerprintingu – cross-site timing. Uživatelé bývají často na internetu v jednu chvíli přihlášení na větším množství webových služeb. K tomu navíc na těchto službách mohou mít na základě své identity přístup k různým zdrojům (například přístup k informacím na profilu nějaké osoby na sociální síti na základě toho, zda je s touto osobou v kontaktu).

Bortz a spol. [85] ukazují, že na základě časové analýzy lze někdy detekovat, zda má uživatel k takovým zdrojům přístup. Podle toho, zda smí uživatel ke zkoumanému zdroji přistoupit, mu totiž bude poskytnuta jiná verze stránky a tedy bude objem stahovaných dat různý. V průměru trvá delší dobu načíst větší množství dat a tedy lze naměřeným časem odhadovat velikost souboru a potažmo tedy i skutečnost, zda má uživatel ke zdroji přístup.

Možnost detekce takto citlivých informací bez vědomí uživatele má samozřejmě širší implikace, pokud jde o bezpečnost a soukromí uživatelů na internetu, nicméně pro účely fingerprintingu lze použít rovněž. Na základě informace o tom, na kterých webových stránkách je uživatel autentizován a k jakým zdrojům má přístup, lze sestavit jeho otisk využitelný pro sledování jeho aktivity.

Útočník pro efektivní provedení časové analýzy však musí být schopen překonat překážku danou poměrně velkým množstvím šumu, který je ovlivněn mimo jiné internetovým připojením a geografickou polohou uživatele [85]. Stejně tak musí být schopen překonat překážky spojené s nepřesným měřením v případě, že bude v prohlížeči uživatele JavaScript vypnutý, například způsobem popsaným v sekci 1.10.1.

1.11 Počet jader v CPU

V současné době je většina uživatelských zařízení (včetně mobilních) připravená na paralelní výpočty větším množstvím jader v procesoru. Informaci o jejich počtu opět poskytne JavaScript – v objektu `navigator` existuje proměnná `hardwareConcurrency` dostupná pouze pro čtení, která udává počet logických jader na používaném zařízení [87].

Význam tohoto API se projevuje především v souvislosti s Web Workers, což jsou JavaScriptové objekty umožňující spouštět kód asynchronně v jiném vláknu, než ve kterém se vykonává hlavní část kódu. Web Workers umožňují paralelizaci složitějších výpočtů a informace o počtu jader tak může posloužit pro jejich lepší plánování [88].

Získat informaci o počtu využívaných jader však lze i v případě, že proměnná `hardwareConcurrency` není v prohlížeči dostupná (například v Internet Exploreru nebo Safari [87]) nebo je nefunkční (v Tor Browseru je z důvodů fingerprintingu `hardwareConcurrency` vždy nastavena na hodnotu 1 [34]). Projekt Core Estimator [89] poskytuje JavaScriptový kód, který počet jader dokáže odhadnout pomocí časové analýzy. Zjednodušený popis jeho postupu je následovný:

1. Změří dobu potřebnou pro vykonání předpřipraveného zdrojového kódu spuštěného v samostatném vláknu pomocí Web Workers API.

2. Spouští zvyšující se počet vláken se stejným kódem jako v bodu 1. Ve chvíli, kdy zaznamená výraznější nárůst času potřebného k vykonání těchto úloh, měření ukončí. Výsledkem je největší počet spuštěných Web Workers, při kterém ještě nedocházelo k neúměrné spotřebě času (což odpovídá situaci, kdy na každém jádru běží právě jedno vlákno).

Většina zařízení, která budou uživatelé využívat k navštěvování webových stránek, budou obvykle vykazovat velmi podobné a vesměs běžné hodnoty počtu jader. Přesto i tuto informaci lze použít jako doplňkovou fingerprintingovou techniku pro větší rozlišení návštěvníků.

1.12 Měření výkonu JavaScriptu

Různé verze prohlížečů podporují odlišné konstrukty v JavaScriptu a jsou různým způsobem optimalizované. To může mít pozorovatelný a měřitelný dopad na rychlost výpočtu. Mowery a spol. [90] provedli experiment, ve kterém se snažili detekovat rozdíly ve výkonu JavaScriptu pro pozdější využití k fingerprintingu.

V tomto experimentu vytvořili 39 různých testů, kterými měřili výkon JavaScriptu se zaměřením na různé charakteristiky jazyka. Výstupem každého měření tedy byl vektor o 39 prvcích, z nichž každý prvek představuje naměřený výkon pro jeden z 39 testů. Tento výstupní vektor byl následně normalizován, aby se co nejvíce zahladily stopy po vnějších vlivech nepodstatných pro experiment (např. výkon CPU, který plošně snižuje čas potřebný pro každý z testů). Jednotlivé instance měření udávají významně odlišné časy, přestože byly spouštěny na stejném stroji i verzi prohlížeče [90].

Na základě dat naměřených pro různé (tehdy běžně používané) verze prohlížečů pak Mowery a spol. [90] zkoušeli klasifikovat další vstupní instance.

Klasifikace probíhala jednoduše tak, že k otisku klasifikovaného prohlížeče byla pomocí Euklidovské vzdálenosti nalezena nejpřesnější shoda s předpočítanými výsledky. Správný odhad prohlížeče tato metoda vydala pro 79.8 % z 1015 vstupních konfigurací (s různými kombinacemi operačních systémů, použitého hardware a samozřejmě verzí prohlížeče). Správného určení „rodiny“ prohlížeče (Chrome, Firefox apod.) dosáhla tato klasifikační metoda dokonce v 98,2 % případů. Autoři následně rozšířili tuto techniku i na detekci operačního systému a procesorové architektury. I zde byli schopni dosáhnout relativně zajímavých výsledků, byť museli podstoupit některé kroky pro stabilizaci prostředí (například pro všechny testy používali zafixovanou verzi prohlížeče Firefox 3.6).

Autoři experimentu se snažili pomocí měření výkonu JavaScriptu rozpoznat verzi prohlížeče nebo jinou charakteristiku prostředí využívaného uživatelem. Je ovšem potřeba zmínit, že pro fingerprinting konkrétního zařízení lze tento krok klasifikace vynechat. Otiskem uživatele nemusí být nutně použitá verze prohlížeče, pro ten účel postačí rovnou výsledky provedených měření, které agregují všechny charakteristiky použitého zařízení dohromady.

Kapitola 2

Obranné techniky a nástroje

Jak jsme ukázali v kapitole 1, způsobů, jakými lze vytvořit otisk prohlížeče a/nebo uživatele, je celá řada a navzájem se liší množstvím informací, které poskytují (entropií), složitostí implementace, technologiemi, které jsou potřeba pro jejich zavedení, i filosofií, s jakou k problému přistupují. Proto není možné zavést jakousi univerzální obranu, která bude chránit proti všem fingerprintingovým technikám najednou. Místo toho je mnohdy potřeba zaměřit se na menší skupinku podproblémů a vypořádavat se tak s technikami fingerprintingu na individuálnější bázi.

2.1 Problémy s implementací obranných technik

V kapitole 1 jsme ukázali několik možných motivací k implementaci fingerprintingových technik. Jak je vidět, ne všechny jsou pro uživatele škodlivé a proto je rozumné v konkrétních případech zvážit, zda je zavedení obran proti fingerprintingu přínosné a žádoucí. Pokud ovšem chceme fingerprintingu předcházet, je potřeba být při implementaci obran ostražitý – jak ukážeme v této sekci, neuvážené zavádění obrany totiž může vést ke ztrátě funkcionality i narušení soukromí uživatele.

2.1.1 Obtížná detekovatelnost

Je náročné rozeznat, zda a jak konkrétní stránka využívá fingerprinting, protože téměř každá detekovatelná vlastnost a chování prohlížeče lze pojmout jako zdroj pro otisk [91] a každé nové API, pro které prohlížeče zavedou podporu, bude pravděpodobně podléhat stejnému nešvaru. Kvůli tomu se nelze spolehnout na jednu konkrétní metodu, jak rozpoznat, zda na stránce dochází k fingerprintingu.

Navíc aktivní fingerprinting pomocí JavaScriptu (který patří mezi nejrozšířenější) lze nejrůznějšími metodami (například obfuskací kódu) maskovat a automatizované odhalení takového chování je proto obtížné – už jen proto, že

nelze snadno rozeznat, zda je snaha o shromáždění informací legitimní, nebo závadná. Možností, jak rozbor přesto provést, je dynamická analýza kódu nebo využití nějaké heuristiky (například doplněk Privacy Badger [92], který se snaží zamezit sledování třetích stran, využívá informace o tom, zda se stejný zdroj načítaný z více různých stránek snaží využívat „podezřelé“ API, jako například Canvas API). Alternativou k detekci fingerprintingu je zavedení „preventivních opatření“, která budou modifikovat chování prohlížeče plošně, tj. bez ohledu na to, zda konkrétní stránka fingerprinting využívá.

2.1.2 Narušení funkcionality

Prvním přirozeným problémem, na který narazíme při návrhu obran proti fingeprintingu, je narušení běžné funkcionality poskytované prohlížečem. Podstatnou část fingerprintingových technik lze obejít jednoduchým vypnutím JavaScriptu (i když jak jsme ukázali v kapitole 1, zdaleka se nejedná o všechny techniky). Problémem takto radikálního přístupu je, že velká část webových stránek přestane být pro uživatele přívětivá a některé úplně přestanou být použitelné (pomineme v tuto chvíli, že skutečnost, že má uživatel vypnutý JavaScript, lze také využít k fingerprintingu).

Analogicky se lze dívat i na další, třeba méně invazivní obranné strategie – drtivá většina z nich vyžaduje pro správné fungování nějaký zásah do běžného chování prohlížeče, což může mít za následek neintuitivní nebo uživatelsky nepohodlné chování stránek nebo dokonce újmu na bezpečnosti (například v případě vypnutí HSTS).

2.1.3 Kontraproduktivní výsledky

Jak uvádí Laperdrix a spol. [93], jednou z největších překážek při zavádění obran je skutečnost, že záměrné přenastavení atributů prohlížeče využívaných k fingerprintingu může paradoxně vést k nechtěnému zvýšení identifikovatelnosti zařízení. To se může projevit například v situaci, kdy vývojáři prohlížeče vydají aktualizaci, která způsobí, že nasazená neaktualizovaná obrana proti fingerprintingu začne navštíveným stránkám vystavovat zastaralé hodnoty. Ty začnou být v novém kontextu velmi ojedinělé a tudíž snadno zneužitelné k vytvoření otisku.

Druhým podstatným faktorem, jak uvádí Eckersley [94], je množství uživatelů, kteří danou obranu zavádí stejným způsobem. Snaha o vyhnutí se fingerprintingu je totiž potenciálně rovněž detekovatelná a může proto sama sloužit jako objekt sledování, a tudíž je-li implementace konkrétní obrany málo rozšířená, sama o sobě může posloužit jako identifikátor.

2.2 Přístupy k obraně

V současné době se rozvíjí několik způsobů, kterými se lze proti sledování bránit. Jak rozebereme v této sekci, v poslední době začaly i široce oblíbené prohlížeče podnikat kroky pro omezení možnosti fingerprintingu, ovšem tato činnost stále nepokrývá celé spektrum způsobů sledování a proto fingerprinting zůstává účinnou technikou. Proto jsou i nadále relevantní i jiné způsoby obran, například pomocí doplňků.

2.2.1 Obrana integrovaná v prohlížeči

Jak uvádí Al-Fannah a spol. [95], ve snaze o redukci fingerprintingu v zájmu zvýšení soukromí uživatelů hrají klíčovou roli tvůrci prohlížečů. Zavádění obran prostřednictvím proprietárních doplňků a filtrování obsahu totiž často trpí většinou problémů popsaných v sekci 2.1.3. Naproti tomu zavedení obran jako součásti prohlížeče, respektive jejich zohlednění v návrzích funkcionalit, které prohlížeče poskytují, těmto nedostatkům předchází:

- Obrany nebudou zastarávat s novými verzemi prohlížeče (naopak se mohou s novými verzemi zlepšovat).
- Obrany jsou tak rozšířené, jak rozšířené jsou prohlížeče, které je implementují, tzn. u populárních prohlížečů nehrozí, že by bylo možné identifikovat uživatele pouze na základě toho, že zavádí nějakou antifingerprintingovou ochranu.

Integrace antifingerprintingových obran přímo v prohlížeči navíc zvyšuje dostupnost a přívětivost využívání těchto služeb i běžnému uživateli a rovněž tak do jisté míry podporuje všeobecnou informovanost a poskytuje kontrolu nad soukromím rozšířenému množství lidí.

2.2.1.1 Sjednocení chování prohlížečů

Rozdíly v chování prohlížečů vytváří velkou plochu pro fingerprinting. Jedním z důvodů je skutečnost, že detaily implementace API jsou typicky ponechány na vývojářích prohlížečů a standardy ukazují ve svých specifikacích pouze vysokoúrovňový pohled – pozorovatelné chování (ovšem jenom to zamýšlené, nikoliv detekovatelné pouze postranními kanály). Al-Fannah a spol. [95] uvádí jako jedno z možných řešení přesnou a detailní specifikaci nejen vnějšího, ale i vnitřního chování implementace API ve standardech – toto opatření by mohlo pomoci eliminovat problémy jako například možnost detekce verze prohlížeče na základě měření výkonu JavaScriptu, jak je popsáno v sekci 1.12.

Významným důsledkem snahy o unifikaci chování prohlížečů je dokument *Mitigating Browser Fingerprinting in Web Specifications* [96] z roku 2019. Ten

uvádí jako příklad problematického chování skutečnost, že seznam systémových fontů lze z pluginů Flash nebo Java Applet získat v nespecifikovaném pořadí, které je závislé na systému uživatele. Kromě informace o tom, jaké fonty uživatel používá, tak lze k fingerprintingu navíc zneužít i informaci o získaném pořadí fontů, a to v podstatě pouze proto, že standard žádné pořadí neudává.

2.2.1.2 Limitace JavaScriptových API

Omezení nebo přímo zákaz některých JavaScriptových API by značnou měrou mohlo přispět ke snížení množství informací, které lze k fingerprintingu zneužít. Snyder a spol. [97] uvádí, že více než 50 % testovaných JavaScriptových API není na nejnavštěvovanějších deseti tisíci stránkách (podle seznamu Alexa Top 10 000) vůbec využíváno. Jejich zákaz by tedy nemusel způsobit závažný negativní zásah do uživatelského komfortu a naopak by mohl přinést podstatné zlepšení situace v oblasti fingerprintingu.

Není-li z nějakého důvodu žádoucí API přímo zrušit, jeho dopady na soukromí lze, jak uvádí Das a spol. [98], snížit dvěma technikami: Obfuskací a snížením přesnosti. Oba přístupy vysvětlíme níže:

- **Obfuskace:** Záměrnou drobnou úpravou hodnot, které API poskytuje, a to ideálně náhodným způsobem, lze docílit eliminace kontinuity mezi jednotlivými naměřenými hodnotami, což v důsledku sice vytvoří uživateli potenciálně unikátní otisky, ale znemožní přiřadit jeden otisk k druhému, neboť se neshodují. Tento přístup využil ve své práci Jiří Sixta [7], když implementoval obranu proti fingerprintingu pomocí Canvas API, ve kterém náhodně drobně upravoval hodnoty barev při vykreslení obrázku.

- **Snížení přesnosti:** Při návrhu API lze definovat legitimní a zamýšlené případy užití a nastavit chování API tak, aby sloužilo pouze těmto případům a neposkytovalo nadbytečné informace, které lze využít pouze pro fingerprinting. V sekci 1.2.1 jsme představili Battery Status API, na kterém lze nedodržení tohoto přístupu dobře demonstrovat. Olejnik a spol. [21] ukázali, že lze toto API zneužít pro měření kapacity baterie, což může posloužit pro fingerprinting – to však je možné pouze kvůli tomu, že API poskytuje zbytečně přesné hodnoty udávající stav baterie. Jako jedno z řešení (které bylo nakonec na jejich doporučení implementováno v prohlížeči Mozilla Firefox) [21] ve své práci sami uvádí zaokrouhlování výstupních hodnot.

2.2.1.3 Tor Browser

Tor Browser je webový prohlížeč vyvíjený neziskovou organizací The Tor Project [99]. Jedná se o prohlížeč, který využívá kód Mozila Firefox a upravuje jej

s cílem zpřístupnění většího soukromí, anonymity a bezpečí svým uživatelům. Samotný Tor Project na svém blogu uvádí, že zhruba 95 % kódu Tor Browseru původně pochází od Firefoxu [100].

Tor Browser usiluje o dosažení soukromí na více frontách, ale obrana proti fingerprintingu je jednou z těch podstatných. V dokumentu *The Design and Implementation of the Tor Browser* [34] hovoří o přístupu a obecných strategiích, které zavádějí, i o konkrétních obranách, které implementují proti současným fingerprintingovým hrozbám (včetně většiny těch, které jsme podrobně rozebrali v kapitole 1).

Tor Browser se snaží získat pro uživatele anonymitu spíše způsobem uniformizace chování prohlížeče [34], tj. mimo jiné snižováním přesnosti hodnot API, zaváděním jednotného seznamu fontů pro všechny uživatele apod. Cílem je vytvoření prostředí, které bude vykazovat podobný otisk pro všechny uživatele Tor Browseru.

Dalším využívaným přístupem je možnost vytvoření „nové identity“ stiskem jednoho tlačítka, kterým lze vymazat všechny identifikátory uživatele a v podstatě v očích webových stránek vytvořit „nového uživatele“. Mimo jiné se tímto tlačítkem vymaže záznam o uložených hodnotách HSTS, aby se zamezilo možnosti existence supercookies, o kterých jsme pojednali v sekci 1.6. Při příští návštěvě se tak ale uživatel stává znovu náchylným na SSL strip a potažmo na útok typu MITM.

2.2.1.4 Mozilla Firefox

Tseng a Barnes [100] uvádí, že Tor Browser úzce spolupracuje s vývojáři Mozilla Firefox a navzájem si pomáhají zvyšovat bezpečí a anonymitu uživatelů. Toto tvrzení je podpořeno skutečností, že Mozilla v posledních letech nasadila ve svém prohlížeči množství opatření proti fingerprintingu, z nichž u některých lze pozorovat kontinuitu s implementací dříve použitou v Tor Browseru. Těchto opatření je celá řada, uvádíme jen některé významnější:

1. Od verze 72 Firefox blokuje všechny požadavky na třetí strany, u kterých vyhodnotil, že provozují fingerprinting a zároveň sledují uživatelskou aktivitu [101]. Využívá k tomu seznam služeb udržovaný společností Disconnect [102], který Mozilla sama pomohla rozšířit. Disconnect udržuje veřejná odůvodnění [103] pro umístění konkrétní služby nebo domény na svůj seznam. V těchto odůvodněních typicky ukazuje útržky kódu, které daná služba využívá pro fingerprinting nebo popisuje přímo celý proces fingerprintingu včetně URL adres, na které je výsledný otisk prohlížeče zasílán.

2. Od verze 58 obsahuje nastavení Firefoxu několik možností konfigurace ohledně chování prohlížeče při setkání s elementem canvas [104]. Lze například nastavit, aby byl uživatel vždy, když se webová stránka pokusí extrahovat data z elementu canvas, prohlížečem upozorněn a dostal

možnost zakázat extrakci v případě, že pojme podezření, že se jedná o fingerprinting.

3. Od verze 55 umožňuje redukci přesnosti měření času v JavaScriptu, čímž chrání proti útokům pomocí časové analýzy [105]. Úroveň zaokrouhlení lze v nastavení prohlížeče upravit.

4. Od verze 52 Firefox umožňuje ochranu proti skenování seznamu fontů – v nastavení lze naplnit proměnnou `font.system.whitelist` seznamem fontů, které jedině smí prohlížeč při zobrazování stránek používat (whitelistem). To může mít samozřejmě negativní dopad na vzhled stránek, nicméně pokud je proměnná naplněna rozumným seznamem fontů, které jsou mezi veřejností běžné, představuje tato funkcionalita vcelku efektivní obranu proti otisku seznamu fontů.

5. S verzí 67 byla nasazená obrana proti fingerprintingu rozlišení okna [106], kterou používá i Tor Browser [107]. Obrana spočívá v přidání okraje po stranách okna, čímž dojde ke zmenšení šířky obsahu na nejbližší nižší násobek 128 pixelů a zmenšení výšky na nejbližší nižší násobek 100 pixelů. Okna prohlížeče s podobnými rozměry tak vykazují naprosto stejné hodnoty rozlišení a nelze je pomocí fingerprintingu odlišit.

Skutečnost, že Firefox aktivně implementuje obrany proti fingerprintingu, je velmi přínosná z pohledu uživatelského soukromí. Ze současných statistik na stránkách The Tor Project [108] lze odhadovat, že Tor Browser má něco přes dva miliony aktivních uživatelů. Přestože toto číslo je úctyhodné, Mozilla Firefox je využívaná (na alespoň mesíční bázi) zhruba 250 miliony uživatelů [109] a představuje v současné době podíl přibližně 4.58 % trhu [110]. To už vytváří silný základ pro umožnění efektu „ztracení se v davu“, tj. zajištění, že detekce implementace antifigerprintingových obran na prohlížeči uživatele sama nepředstavuje dostatečně silný otisk a tedy nepovede ke kontraproduktivním výsledkům (jak jsme rozebrali v sekci 2.1.3).

Jistý problém představuje fakt, že část antifingerprintingových obran není ve Firefoxu aktivovaná ve výchozím módu nebo jsou tyto obrany kvůli snaze o kompromis s uživatelským komfortem aktivované pouze v omezené míře, a představují tak pouze částečnou ochranu. Samozřejmě lze očekávat, že velké množství uživatelů povětšinou nechá nastavení prohlížeče v původním stavu a nebude tak čerpat všechny výhody, které prohlížeč může poskytnout – důsledkem je bohužel částečná ztráta síly těchto opatření i pro ty uživatele, kteří se rozhodli obrany aktivovat v plném rozsahu (opět z důvodů uvedených v sekci 2.1.3).

2.2.1.5 Google Chrome

Mozilla Firefox proaktivně pečuje o soukromí uživatelů, což je podpořeno i kooperací s Tor Browserem. Faktem ale je, že v současné době dominuje trhu

Google Chrome (podle [110] ho využívá 64,45 % uživatelů). Proto je obzvlášť signifikantní, jaký přístup k problematice fingerprintingu zaujímá právě tento prohlížeč.

Chrome roku 2019 uveřejnil svůj plán pro další postup ve vývoji prohlížeče pokud jde o soukromí uživatelů [111]. Uvádí v něm, že blokování cookies znemožňuje vývojářům identifikovat uživatele (například pro účely marketingu) a užívání fingerprintingových technik je důsledkem snahy najít stabilnější alternativu. Navíc uvádí, že plošné omezení sledování uživatelů bez poskytnutí alternativy by způsobilo menší efektivitu reklam, což by mohlo vést ke snížení množství volně dostupného obsahu. Chrome proto plánuje podniknout v oblasti soukromí uživatelů mimo jiné tyto podstatné kroky:

- **Podpořit používání cookies**, aby se provozovatelé stránek k fingerprintingu uchylovali méně [111]. V rámci této podpory chtějí zavést například klasifikaci cookies, aby bylo odlišeno, které z nich jsou používané pro účely funkcionality stránek (například udržování relace přihlášení) a které jsou používáné pro sledování uživatele [112]. Cílem je uživatelům přinést větší přehled a kontrolu nad tím, kdo je sleduje a v jaké míře.

- **Redukovat možnosti fingerprintingu**, tj. začít přímo podnikat kroky, které budou fingerprinting prohlížeče omezovat. Chromium Blog [112] uvádí, že Chrome chce snížit počet způsobů, jak lze vytvořit otisk prohlížeče pomocí pasivního fingerprintingu (například redukcí informací obsažených v hlavičce User-Agent [113]), a detekovat aktivní fingerprinting a zasahovat proti němu. Chrome chce rovněž zavést redukci informací poskytovaných skrze Battery Status API a informace o senzorech zařízení (například akcelerometr, o jehož zneužití pro fingerprinting jsme hovořili v sekci 1.9.2) [113].

- **Odstranit podporu pro cookies třetích stran** [114] v horizontu dvou let. V rámci procesu ukončování podpory těchto cookies chce Google vytvořit mechanizmus, který je z pohledu marketingových agentur nahradí téměř plnohodnotně, ale přitom bude transparentnější pro uživatele a bude jim poskytovat více soukromí. Tento mechanizmus by měl být schopen doručit reklamu cílové skupině navzájem podobných lidí, aniž by identifikující data o těchto lidech opustila jejich webový prohlížeč (tzn. informace o uživateli by měly být třetí straně odhaleny, až když se ověří, že uživatel na základě těchto údajů nemůže být jednoznačně identifikován) [115]. Pro tento účel je v současné době vyvíjeno nové API, tzv. *Aggregated Reporting API* [116], které by mělo být založené na podobné technologii jako `localStorage`.

Jak jsme ukázali v kapitole 1, vývojáři Google Chrome postupně zavádí antifingerprintingová opatření (zrušení podpory pro Adobe Flash, úprava chování HSTS, zrušení podpory SDCH atd.) a na základě jejich výroků [111] se

lze domnívat, že mají v plánu v této činnosti pokračovat. Na druhou stranu faktem je, že množství obran, které byly implementovány a těch, které ještě mají v plánu implementovat [113], je stále o poznání užší, než množství obran, které jsou v současné době už nasazené v prohlížeči Mozilla Firefox nebo Tor Browser.

Přístup k boji s fingerprintingem prostřednictvím rozšíření podpory cookies se nicméně může ukázat jako efektivní. Je ovšem otázka, zda rozšíření možností, jak může uživatel nakládat se svými cookies, nebude mít spíše opačný efekt. Větší uživatelská kontrola totiž může vést k tomu, že budou cookies méně spolehlivým nástrojem pro sledování uživatelů a provozovatelé stránek tak budou využívat fingerprinting ještě více.

2.2.2 Zavedení univerzálního identifikátoru

Al-Fannah a spol. [95] uvádí, že tvůrci webových prohlížečů nemusí mít zájem o plošnou eliminaci fingerprintingu, neboť v některých případech sami fingerprinting využívají. Al-Fannah a spol. [117] ukázali, že mezi deseti tisíci nejnavštěvovanějšími webovými stránkami (podle seznamu milionu nejnavštěvovanějších stránek *The Majestic Million* [118]) je Google Analytics nejčastější třetí stranou, která využívá fingerprinting [119] (přičemž Google Chrome používá podle současných statistik [110] více než 64 % trhu).

Dá se tedy říci, že je v zájmu společností poskytujících službu personalizované reklamy zachovat nějaký způsob, jak rozpoznat uživatele na internetu, a fingerprinting má v tomto směru atraktivní vlastnost, kterou běžné cookies postrádají – uživatel se totiž svého otisku nemůže snadno zbavit. Al-Fannah a spol. [95] se ve svém návrhu řešení podobně jako Google Chrome [111] ubírají cestou eliminace fingerprintingu vytvořením silnějšího nástroje pro sledování uživatelů. Namísto podpory cookies však navrhují jako možný přístup zavedení *univerzálního identifikátoru*.

Tento identifikátor by byl generován prohlížečem a prohlížeč by se jím prokazoval na každé navštívené webové stránce. Jak cookies, tak i fingerprinting by pak mohly ztratit význam, neboť uživatel by byl identifikovatelný za každých okolností a relace session jednotlivých serverů by se vztahovala přímo k tomuto identifikátoru [95]. Využití tohoto identifikátoru by pak mělo být menší hrozbou pro soukromí uživatelů než fingerprinting, neboť identifikátor sám o sobě (narozdíl od fingerprintingu) funguje pro uživatele zcela transparentně a navíc neposkytuje serveru žádnou nadbytečnou informaci o uživateli, pouze udává jeho totožnost.

Práce [95] dále uvažuje potenciální výhody tohoto systému; prohlížeče by mohly umožnit uživatelům „restartovat“ svůj identifikátor, tj. v podstatě vygenerovat novou identitu (což by byl silnější ekvivalent současného mazání cookies). Rovněž by mohlo být podporováno udržování více identifikátorů současně – například jeden pro autentizaci, druhý pro personalizované reklamy,

případně více identifikátorů jako ekvivalent vytvoření více identit pro různé situace a potřeby.

Proti implementaci tohoto přístupu a vytěžení jeho výhod však stojí dvě překážky:

- Bylo by potřeba zajistit, aby navrhovaný identifikátor představoval alespoň ekvivalentně silný nástroj pro sledování uživatelů, jaký v současnosti představuje fingerprinting. To ale nelze zaručit. Jak jsme ukázali v kapitole 1, některé fingerprintingové techniky poskytují možnosti pro sledování uživatelů dokonce napříč zařízeními a zaručují, že otisk nelze ze strany uživatele snadno restartovat, takže je uživatel potenciálně identifikovatelný i tehdy, když vynaložil úsilí na zastření svojí identity. Tuto míru využitelnosti univerzální identifikátor nedokáže poskytnout – uživatelé by podle návrhu měli možnost měnit identifikátor přímo v prohlížeči, a i kdyby tomu tak nebylo, jednoduchý filtr na uživatelském proxy serveru (jako například v Privoxy) by pravděpodobně byl schopen tuto službu bez problému zastat.

- I pokud by se podařilo zavést identifikaci pomocí ID prohlížeče, neexistuje záruka, že webové stránky nebudou nadále používat fingerprinting, ať už z jakéhokoliv důvodu. Cílem implementace identifikátoru je umožnit vývojářům prohlížečů, aby začali proaktivně zavádět antifingerprintingové obrany – ne každému druhu vytváření otisku však lze zamezit bez výrazné újmy na použitelnosti nebo bezpečnosti. Identifikátor prohlížeče by se tak v důsledku mohl stát pouze jedním dalším objektem, který lze zneužít k fingerprintingu a přinesl by tak pouze usnadnění sledování uživatelů bez kýženého přínosu v podobě uživatelské kontroly a částečného navrácení soukromí. Al-Fannah a spol. [95] navrhují, že by bylo možné překonat tuto překážku například regulacemi, ovšem je otázka, do jaké míry je to realistické nebo vůbec žádoucí.

2.2.3 Obrana implementovaná v doplňcích

Jedním z oblíbených přístupů k zavádění antifingerprintingových obran je delegace této činnosti samostatnému řešení v podobě doplňku nebo jiné aplikace, která rozšiřuje běžné chování prohlížeče. Při jejich nasazení je potřeba ostražitost, neboť jejich obrany (na rozdíl od obran implementovaných přímo v prohlížeči) mohou být nestandardní a tedy vytvořit větší prostor pro identifikaci uživatele.

Doplňků, které mají za cíl zvýšit soukromí uživatelů, je nepřeberné množství a přístupy k implementaci obran se mezi jednotlivými řešeními liší. Například NoScript [120] přistupuje k obraně agresivním způsobem tak, že vypne veškeré skripty (JavaScript, Java, Flash a další) u všech domén, které nejsou uvedené na seznamu důvěryhodných. Podstatná část doplňků implementuje

obranu proti jedné fingerprintingové technice nebo myšlenkově blízké podmnožině fingerprintingových technik (například Canvas Blocker [121] poskytuje obranu proti zneužití některých JavaScriptových vlastností, jako např. Canvas API, Audio API, objekt `navigator` a další). Většina doplňků preventivně upravuje chování prohlížeče tak, aby byl fingerprinting obtížný, jiné (například Privacy Badger [92]) využívají heuristik k tomu, aby fingerprinting rozpoznávaly a eliminovaly ho až následně. Podrobnější informace o větším množství používaných doplňků v poslední době agreguje například Laperdrix a spol. [93].

Jedním ze způsobů, jak upravit chování prohlížeče ve snaze chránit uživatele před fingerprintingem, je zavést filtrování obsahu na úrovni proxy serveru. To je také jedním z cílů této práce. Proxy serveru Privoxy a implementacím zvolených obran se budeme věnovat v následujících kapitolách.

2.2.4 Vyhodnocení účinnosti obran

Na internetu je volně dostupné množství nástrojů, které umožňují uživatelům ověřit si účinnost obran, které proti fingerprintingu nasazují. Mezi nejpoužívanější a často skloňované patří nástroje Panopticlick [64] a AmIUnique [65]. Tyto nástroje umožňují uživatelům prohlédnout si svůj otisk (vytvořený pomocí vybraných fingerprintingových technik) a také zjistit míru identifikovatelnosti jednotlivých atributů i svého zařízení jako celku. Identifikovatelnost je však stanovena pouze na základě dat sesbíraných od uživatelů těchto služeb (kteří kromě jiného v disproporční míře nasazují antifingerprintingové obrany), což může její realističnost do velké míry zkreslovat [122].

Přestože je míra identifikovatelnosti udávaná v nástrojích AmIUnique a Panopticlick pouze přibližná, poskytuje i tak uživateli přidanou hodnotu v podobě orientační informace o jeho anonymitě. Na druhou stranu seznam používaných fingerprintingových technik není v těchto nástrojích zdaleka úplný, a pro otestování méně obvyklých obran je proto potřeba zvolit jiné nástroje. Dostupné jsou jednoduché nástroje, které uživateli demonstrují jednu konkrétní fingerprintingovou techniku. Příkladem budiž nástroj HSTS Super Cookie [123], který vytvoří a zobrazí uživateli jeho šestnáctibitový otisk vytvořený pomocí HSTS.

Nástroj Browserleaks [124] umožňuje uživateli určit svůj otisk vytvořený pomocí některých JavaScriptových API (WebRTC, Canvas, Web Audio, Battery Status a další) i některých technik využívajících kaskádové styly (rozlišení a orientace obrazovky, podpora nestandardních CSS vlastností a další).

Podobně i nástroj Browserize [125] implementuje větší množství fingerprintingových metod a umožňuje uživateli ověřit svůj otisk. Nástroj obsahuje kromě běžnějších technik (Canvas, WebRTC a další) i méně obvyklé techniky jako fingerprinting fontů pomocí CSS nebo otisk pomocí hlavičky ETag. U každé fingerprintingové techniky je navíc uvedeno stručné pojednání o funkcionalitě a některé relevantní zdroje pro přiblížení problematiky uživateli.

Kapitola 3

Analýza obran navržených pro Privoxy

Privoxy [5] je proxy server, který filtruje a modifikuje webový obsah za účelem zvýšení soukromí a odstranění nežádoucího nebo obtěžujícího obsahu. Jedná se o svobodný software, licence tohoto projektu je GNU GPLv2. Zdrojový kód je psaný v jazyku C.

3.1 Obrany implementované v původním kódu Privoxy

Privoxy implementuje několik vlastních opatření, která slouží k ochraně soukromí uživatele. Několik z nich lze při správném uživatelském nastavení s výhodou využít i proti fingerprintingu. V této sekci popíšeme jejich chování na základě analýzy zdrojových kódů Privoxy.

3.1.1 Hlavičky pro manipulaci s cache

V sekci 1.3 jsme nastínili, jak se dá zneužít hlavička `Last-Modified` k vytvoření supercookie. Privoxy zavádí obranu, která lze aktivovat direktivou `overwrite-last-modified`. Tato obrana má následující průběh:

1. Pokud uživatel předal direktivě parametr `block`, hlavička se jednoduše vymaže dříve, než dorazí do prohlížeče a algoritmus končí.
2. Pokud uživatel předal direktivě parametr `reset-to-request-time`, hlavička se nastaví na aktuální čas a algoritmus končí.
3. Pokud nelze parsovat čas určený hlavičkou, hlavička se vymaže (toto chování není u prohlížečů standardem, což vytvoření supercookie značně usnadňuje).

4. K času v hlavičce se připočte drobná, náhodně zvolená kladná odchylka. Tato odchylka musí být dostatečně malá, aby výsledek neukazoval čas z budoucnosti. Cílem přičtení této odchylky je porušit případný otisk zakódovaný do platného data.

Zajímavý přístup zvolili autoři Privoxy k řešení situace, kdy hlavička `Last-Modified` odkazuje na časový údaj, který je z budoucnosti. Takový čas nemá ze sémantického pohledu smysl, neboť hlavička má ukazovat čas poslední úpravy zdroje [126]. Možným řešením této situace by bylo hlavičku zahodit jako neplatnou (podobný přístup byl zvolen v kroku 3 pro časy, které nelze zpracovat). Namísto toho však Privoxy hlavičku zachová a od času (narozdíl od běžného postupu, který je popsaný v kroku 4) odečte drobnou kladnou odchylku.

Analogický systém je využit i v direktivě `hide-if-modified-since`, která aplikuje podobné obrany na hlavičku `If-Modified-Since` v odchozích požadavcích.

Filtrování `Last-Modified` umožňuje fakt, že má tato hlavička předepsaný formát, který lze ověřit a v případě jehož nesplnění lze adekvátně zakročit. Hodnota hlavičky `ETag`, která lze ke sledování uživatele použít rovněž, však žádnou stanovenou syntax nemá a její filtrování (bez újmy na funkcionalitě cache) je prakticky nemožné. Privoxy proto poskytuje pouze možnost ze všech odchozích požadavků úplně odstranit hlavičku `If-None-Match`, která zasílá serveru zpět ETag dokumentu. Tato obrana lze spustit pomocí direktivy `crunch-if-none-match`.

3.1.2 Další hlavičky

Privoxy poskytuje několik dalších jednoduchých obran proti fingerprintingu, který pro vytváření otisku zneužívá HTTP hlaviček.

- K vymazání hlavičky `Accept-Language`, případně ke změně její hodnoty, slouží direktiva `hide-accept-language`. Její využití může pochopitelně přinést pro uživatele podstatné komplikace. Změna hodnoty například na jazyk, který uživatel nezná, může vést k nečitelnosti obsahu.
- Pro úpravu hlavičky `User-Agent` slouží direktiva `hide-user-agent`. Direktiva přijímá jako parametr řetězec, který obsahuje informaci o verzi prohlížeče, kterou se chce uživatel prokazovat. V rámci práce [7] bylo této hlavičky využito v kombinaci s vlastním kódem pro zakrytí verze prohlížeče v JavaScriptu.

3.2 Rozšíření proti fingerprintingu

V rámci práce [7], na kterou navazujeme, bylo vytvořeno rozšíření Privoxy verze 3.0.26, které zapouzdřuje obrany proti fingerprintingu. V této sekci shr-

neme informace, které jsou relevantní pro navazující práci. Rozšíření obsahuje následující antifingerprintingová opatření:

- **Maskování použitého prohlížeče**, ve kterém se pomocí úpravy atributů objektu `navigator` přepisuje informace o použité verzi a modelu prohlížeče podle přání uživatele.
- **Úprava chování elementu canvas** tak, aby nebylo možné vytvořit s jeho pomocí otisk. Volání funkcí `toDataURL`, `toBlob` a `getImageData` je v navštívených stránkách rozšířeno o volání vlastního JavaScriptového kódu funkce, který chování původní funkce pozměňuje. Uživatel si může vybrat, zda bude funkce vracet data odpovídající prázdné instanci canvasu, nebo zda se před čtením náhodně mírně pozmění barvy pixelů na stávajícím canvasu, čímž se otisk „znehodnotí“.

3.2.1 Využité prostředky

Bylo snahou původního rozšíření při implementaci co nejméně zasahovat do původního kódu. Přesto však nebylo využito direktivy `external-filter`, pomocí které je možné předat obsah ke zpracování externímu programu a potom ho načíst zpět z výstupu tohoto programu – důvody byly následující [7]:

- S externími filtry je bohužel spojená vysoká režie, která využitelnost výsledného řešení snižuje.
- Externí filtry jsou spouštěné se stejnými právy jako Privoxy a obsah určený k filtrování je dočasně ukládán do souboru, což představuje jisté bezpečností riziko.
- Privoxy pracuje s větším množstvím užitečných informací, které nejsou pro externí filtry dostupné.

Implementace obran je tedy realizována kombinací regulárních výrazů (které jsou odděleny v samostatném souboru, aby je mohl uživatel měnit bez nutnosti kompilace projektu) a nového kódu v jazyku C.

3.2.2 Napojení funkcionality na kód Privoxy

Drtivá většina přidané funkcionality je umístěna v souboru `fpblock.c`. Jednotlivé funkce reprezentující konkrétní obrany (tedy `replace_user_agent` pro maskování prohlížeče a `modify_canvas_behavior` pro modifikaci chování canvasu) se volají z ústřední funkce `apply_incoming_fingerprint_defenses`.

Volání funkce `apply_incoming_fingerprint_defenses` je patřičně umístěno do těla funkce `execute_content_filters`, což je původní funkce Privoxy, která má na starost filtrování a modifikaci těl odpovědí, které dorazí směrem ze serveru na klienta. Toto volání, které je v kódu podmíněné tím, že

potenciálním filtrovaným obsahem je textový soubor, představuje hlavní pojítko mezi původním kódem a vytvořeným rozšířením o antifingerprintingové obrany.

Privoxy na základě hlavičky `Content-Type` rozlišuje typ přijaté odpovědi. Pro filtrování je zajímavý především textový obsah, který je označen příznakem `CT_TEXT`. Kvůli jemnějšímu rozlišení typu obsahu byly doplněny nové příznaky:

- `CT_JS` pro JavaScriptové soubory (hlavička `Content-Type` obsahuje podřetězec `javascript`)
- `CT_HTML` pro HTML dokumenty (hlavička `Content-Type` obsahuje podřetězec `text/html` nebo `/xhtml`)

Tyto nové příznaky jsou využity k rozpoznání toho, zda má být použit nějaký filtr (například nemá smysl uplatňovat filtry pro upravení chování canvasu na CSS soubor). Zároveň lze také podle typu souboru rozhodnout, který filtr bude spuštěn (HTML soubory mohou obsahovat JavaScriptový kód, ale obsahují i jiný text, a proto jsou pro stejnou úpravu obsahu definovány mírně odlišné filtry na základě typu souboru).

3.2.3 Seznam akcí a direktiva fingerprint-filter

O tom, zda se jednotlivé obrany budou aplikovat na příchozí odpovědi (případně i na které), může uživatel rozhodnout pomocí umístění příslušné direktivy v souboru `fingerprint.action`, který je evidován v konfiguračním souboru `config` jako vstupní soubor s kolekcí akcí na upravení obsahu. Příklad zápisu direktivy pro modifikaci obsahu si můžeme prohlédnout na výpisu kódu 3.1.

```
1 # Apply canvas randomization to all requests
2 {+fingerprint-filter{canvas-random-all}}
3 /
```

Výpis kódu 3.1: Syntaxe direktivy pro aktivování obrany proti fingerprintingu pomocí canvasu

Direktiva `fingerprint-filter` je zaregistrovaná v souboru `actionlist.h` a přijímá jako parametr název fingerprintingové obrany, kterou si uživatel přeje aplikovat (v případě výpisu kódu 3.1 tedy například `canvas-random-all`). Podle sufixu parametru (`-all` nebo `-js`) může uživatel rozhodnout, zda si přeje obrany aktivovat pouze na JavaScriptové soubory nebo i na všechny HTML soubory.

Kapitola 4

Návrh a implementace

V kapitole 3 jsme popsali obecný přístup k rozšíření funkcionality Privoxy v rámci práce [7] i konkrétní úpravy, které byly do kódu zaneseny. V této navazující práci jsme respektovali tuto strukturu, nicméně kvůli povaze implementovaných obran bylo potřeba do původního kódu Privoxy přidat několik dalších změn.

Rozšíření Privoxy s antifingerprintingovými technikami bylo nasazeno na tehdejší aktuální verzi 3.0.26. Současná verze Privoxy je však 3.0.28, proto bylo prvním krokem přenesení úprav pro fingerprinting do aktuální verze. Díky tomu, že tyto úpravy byly už z návrhu relativně neinvazivní, nedošlo k žádným konfliktům a migrace proběhla bez větších problémů.

4.1 Výběr obran k implementaci

Při výběru obran pro implementaci jsme se zaměřili na ty fingerprintingové techniky, pro jejichž potírání nabízí technologie proxy serveru nejlepší podmínky. Otisk pomocí JavaScriptových API patří mezi nejrozšířenější a nejpotentnější metody fingerprintingu. Na druhou stranu proxy server, jakým je Privoxy, není ideální platformou pro obranu proti nim. Jak jsme nastínili v sekci 2.1.1, minifikace, obfuskace (nebo potenciálně i jiná transformace) zdrojového kódu v JavaScriptu může vést k zamaskování fingerprintingu a přímočará obrana pomocí filtru v takové situaci přestává být účinná.

Proxy server je však dostatečným nástrojem v případech, kdy k implementaci obrany stačí statická analýza kódu. To platí například v případě fingerprintingu pomocí hlaviček nebo pomocí kaskádových stylů (možnosti obfuskace CSS souborů jsou oproti JavaScriptu velice omezené). V této práci jsou tedy implementované obrany proti následujícím technikám:

- Otisk seznamu fontů pomocí CSS, který byl rozebrán v sekci 1.7.1.
- Otisk rozlišení obrazovky pomocí CSS, který byl rozebrán v sekci 1.7.2.

- Otisk orientace obrazovky pomocí CSS, který byl rozebrán v sekci 1.7.3.
- Supercookies pomocí hlavičky HSTS, které byly rozebrány v sekci 1.6.
- Supercookies pomocí hlaviček ETag a Last-Modified, které byly rozebrány v sekci 1.3.

4.2 Nutné úpravy kódy

Přestože této skutečnosti nebylo v práci [7] využito, akce mající na starost fingerprinting jsou definovány pomocí makra `DEFINE_ACTION_MULTI`, což umožňuje k nasazení obrany předat více než jeden parametr[3]. Tohoto faktu využijeme k parametrizaci obran a umožnění větší kontroly uživatele nad chováním spouštěných transformací.

4.2.1 Úpravy kódu pro obrany nad CSS

Pro odlišení typu souboru byl kromě příznaků `CT_JS` a `CT_HTML` vytvořen nový příznak `CT_CSS`, který reprezentuje soubory kaskádových stylů. Tento příznak je dostupný pro všechny funkce, které implementují antifingerprintingové obrany v souboru `fpblock.c`. `CT_CSS` je u odpovědi nastaveno v případě, že hlavička `Content-Type` obsahuje podřetězec `"text/css"`. Proces aktivování příznaku zajišťuje funkce `server_content_type` v souboru `parsers.c`, stejně jako v případě příznaků `CT_JS` a `CT_HTML`.

Uživatel může při aktivaci obrany proti fingerprintingu pomocí canvasu nebo objektu `navigator` v direktivě akce specifikovat, zda chce filtrování provádět pouze na JavaScriptové soubory, nebo i na HTML soubory. Podobnou kontrolu jsme v naší práci uživateli umožnili i v případě obran nad CSS. Příponou `-css` za koncem názvu obrany může uživatel definovat, že si přeje obranu aktivovat pouze na soubory kaskádových stylů. S příponou `-all` se pak filtrování provede v CSS i HTML souborech.

Pro HTML soubory jsou stanoveny mírně odlišné filtry – kontroluje se v nich, zda je filtrovaný kód uzavřen mezi tagy `<style>` a `</style>`. Tím se předchází chybnému filtrování běžného obsahu na HTML stránkách.

4.2.2 Úpravy kódu pro obrany nad hlavičkami

Jak jsme uvedli v sekci 3.2.2, antifingerprintingové obrany jsou napojeny na kód Privoxy voláním z původní funkce `execute_content_filters`. Tuto implementaci bylo potřeba rozšířit kvůli filtrování hlaviček. HTTP hlavičky každé filtrované odpovědi serveru jsou sice parametrem předávány do ústřední funkce `apply_incoming_fingerprint_defenses`, nicméně k volání této funkce

[3] Prvním parametrem je vždy jméno zvolené obrany s příslušným sufixem pro specifikaci druhu souboru, nad kterým je obrana spouštěna.

dojde pouze v případě, že filtrování obsahu odpovědi dává smysl. K tomu musejí být splněny následující podmínky:

1. Uživatel vytvořil v souboru akcí alespoň jednu akci, která odkazuje na některou fingerprintingovou obranu.

2. Odpověď serveru obsahuje nějaké tělo (odpovědi běžně tělo obsahovat nemusí například když se jedná o přesměrování).

3. Odpověď serveru je textová (má příznak `CT_TEXT`).

Volání původní ústřední funkce zůstalo zachováno pro obrany nad *obsahem* odpovědí, ale pro fingerprinting hlaviček byla v souboru `fpblock.c` zavedena nová funkce `apply_fingerprinting_header_filters`, pro jejíž volání není nutné splnění výše uvedených podmínek 2 a 3.

Definice funkce `handle_established_connection` je umístěna v hlavním souboru Privoxy `jcc.c`. Tato funkce má, jak napovídá její název, na starost veškerou výměnu dat mezi klientem a serverem, jakmile se podaří navázat mezi nimi spojení [4]. Právě do této funkce je umístěno filtrování hlaviček prostřednictvím volání `apply_fingerprinting_header_filters`. Tato filtrace je zároveň poslední modifikací hlaviček předtím, než se pomocí funkce `list_to_text` zpracují do textové podoby, aby byly následně odeslány v rámci modifikované odpovědi klientovi.

4.3 Implementace obran

Funkce, které mají na starost jednotlivé antifingerprintingové obrany, jsou umístěny společně s většinou kódu v souboru `fpblock.c`. Na úvod poznamenejme, že cílem implementovaných obran je především zajistit změnu otisku prohlížeče uživatele. Pokud si uživatel rovněž přeje získat méně unikátní otisk, je potřeba správně nastavit parametry a obrany aktivovat s rozvahou – zvláštní důraz by přitom měl být kladen na to, aby informace zakryté pomocí implementovaných obran neunikaly jiným kanálem.

4.3.1 Obrana proti otisku fontů pomocí CSS

Jak jsme rozebrali v sekci 1.7.1, funkcionalita CSS lze zneužít ke zjištění seznamu fontů prakticky ve stejném rozsahu jako například JavaScript. Navrženou obranou je randomizace pořadí fontů ve vlastnosti `font-family`.

Implementace zajišťovaná funkcí `randomize_font_order` prohledává filtrovaný dokument a pomocí regulárního výrazu se snaží najít výskyt CSS vlastnosti `font-family`. Z výrazu potom extrahuje názvy jednotlivých fontů

[4] Není proto překvapivé, že tato funkce obsahuje například i volání kódu pro filtrování obsahu odpovědí.

(předpokládá se, že názvy fontů jsou odděleny čárkami) a pomocí Fisher-Yatesova algoritmu [127] změní jejich pořadí. Jakým způsobem může tato obrana zafungovat na konkrétním příkladu si můžeme prohlédnout na výpisu kódu 4.1.

```
1  //kod pred aplikaci obrany
2  #detection_element {
3    font-family : "Comic Sans", fingerprintFont;
4  }
5
6  #legitimate_element {
7    font-family : font1, font2, font3;
8  }
9
10 //mozny vystup po aplikaci obrany
11 #detection_element {
12   font-family : fingerprintFont, "Comic Sans";
13 }
14
15 #legitimate_element {
16   font-family : font3, font1, font2;
17 }
```

Výpis kódu 4.1: Ukázka možného vstupu a výstupu obrany proti fingerprintingu pomocí fontů

Analogickým způsobem dochází i k randomizaci pořadí jednotlivých záznamů zdroje fontu v pravidle `@font-face`. Tím je zajištěna obrana proti oběma možným implementacím téže fingerprintingové techniky.

4.3.1.1 Dopady nasazení obrany na chování stránek

Takto navržená obrana samozřejmě postihuje i legitimní použití `@font-face` a bude tedy mít negativní dopad na vzhled navštívených stránek. Výhodou ovšem je, že fonty, jejichž pořadí bylo randomizováno (například ve výpisu kódu 4.1 fonty `font1`, `font2` a `font3` v elementu `#legitimate_element`), byly nastaveny vývojářem stránek, a tudíž by všechny měly v daném kontextu poskytovat dostatečnou funkčnost i vzhled. Míra újmy na legitimních použitích je tedy tímto způsobem limitovaná.

Naproti tomu dopad na využití `@font-face` k fingerprintingu je značný. Ve výpisu kódu 4.1 je v elementu `#detection_element` prohozeno pořadí fontů, což povede ke skutečnosti, že prohlížeč se nejprve pokusí vykreslit font `fingerprintFont`. Pokud se jeho prostřednictvím snažila webová stránka zjistit přítomnost fontu `"Comic Sans"`, bude server po prohození informován, že tento font na uživatelově zařízení nainstalovaný není (přestože to nemusí být pravda). Při skenování většího množství fontů tak je výsledný otisk narušen a server dostává zkreslenou informaci, podle které uživatel nemusí být při další návštěvě identifikovatelný.

4.3.1.2 Diskuze nedostatků

Fonty, které uživatel skutečně nemá na zařízení nainstalované, budou serveru reportovány vždy jako nepřítomné, bez ohledu na to, zda je tato obrana aktivní. Otisk tedy může být narušen pouze v případě fontů, které nainstalované jsou. Server by tak teoreticky mohl pomocí statistického vyhodnocení většího množství požadavků od uživatele usuzovat na skutečný seznam nainstalovaných fontů a tuto informaci použít rovněž pro fingerprinting (byť s menší spolehlivostí).

Taková analýza by však vyžadovala větší množství prostředků (otisk uživatele by se musel ukládat při každém požadavku). Navíc by byla potřeba další nasazená funkční fingerprintingová technika, která sama o sobě postačí jako identifikátor uživatele – bez ní není možné jednotlivé požadavky uživatele spárovat a tedy ani provést jejich potřebnou analýzu. Fingerprinting fontů by ale v takové situaci byl redundantní. Z toho důvodu jsou reálné dopady takového rizika omezené.

Dalším problémem je skutečnost, že identifikátory fontů vytvořené v pravidle `@font-face`, jsou-li ohraničeny uvozovkami, mohou obsahovat celou řadu nealfanumerických znaků včetně čárek [128]. Vzhledem k tomu, že algoritmus používá čárku jako oddělovač fontů, může tato skutečnost vést k nesprávnému vykreslení stránek, pokud jsou v CSS kódu používány fonty se jmény, která obsaují čárku v názvu. Na druhou stranu tato situace je velmi okrajová a rozhodli jsme se ji v algoritmu neřešit. Její ošetření by totiž nebylo možné zajistit pouze pomocí regulárních výrazů, což by přineslo relativně velkou režii a zamezilo možnosti použití filtrů.

4.3.2 Obrana proti otisku rozlišení pomocí CSS

V sekci 1.7.2 jsme rozebrali možnosti, jak detekovat rozlišení obrazovky uživatele pomocí kaskádových stylů teoreticky s přesností na jeden pixel. Navržená obrana je inspirovaná podobnou obranou zvanou *letterboxing*, kterou jsme popsali v sekci 2.2.1.4.

Obrana spočívá v tom, že je změněna hodnota v CSS pravidlech `@media`, která určují vzhled stránky na základě velikosti obrazovky nebo okna (například `min-width`). Konkrétně dochází k „zaokrouhlení“ hodnot výšky a šířky na násobek uživatelem zvolené hodnoty. Každý původní rozměr šířky W_{old} je při vstupním parametru P nahrazen novým rozměrem W_{new}, který je definován následujícím předpisem:

$$W_{new} = W_{old} - (W_{old} \bmod P)$$

Pro lepší představu si můžeme prohlédnout příklad vstupního a výstupního kódu algoritmu pro parametr šířky 200 na výpisu kódu 4.2.

V našem příkladu je původní šířka 1024 pixelů zaokrouhlena na nejbližší menší násobek hodnoty 200, což odpovídá výstupní hodnotě 1000 pixelů. Ana-

logickým způsobem probíhá i transformace výšky, pro niž je vyčleněn vlastní parametr a je tedy tímto způsobem škálovaná zvlášť.

```
1  //kod pred aplikaci obrany
2  @media (min-width: 1024px) {
3    #element::after {
4      content: url("/fingerprinting/width?width=1024px");
5    }
6  }
7
8  //vystup po aplikaci obrany
9  @media (min-width: 1000px) {
10   #element::after {
11     content: url("/fingerprinting/width?width=1024px");
12   }
13 }
```

Výpis kódu 4.2: Ukázka možného vstupu a výstupu obrany proti fingerprintingu pomocí detekce rozlišení obrazovky

4.3.2.1 Dopady nasazení obrany na chování prohlížeče

Cílem této obrany je zajistit, aby rozměry obrazovky (případně okna prohlížeče) nevykazovaly příliš unikátní hodnoty. Jelikož technika letterboxing v prohlížeči Mozilla Firefox používá hodnoty parametrů 128 pixelů pro šířku a 100 pixelů pro výšku, je výhodné použít stejné hodnoty jako vstup naší obrany. Přesnost fingerprintingu rozlišení se pomocí této techniky významně snižuje, což uživateli umožňuje „ztratit se v davu“.

Chování běžných webových stránek, které fingerprinting neprovozují, bude pozměněno rovněž. Aplikace výše popsaných transformací způsobí, že se budou na webovou stránku aplikovat některé kaskádové styly, u kterých to tvůrci nezamýšleli (a naopak nebudou aktivní některé styly, které by měly být). Vzhled stránek tak může být uzpůsoben jinému než uživatelovu skutečnému rozlišení. Na druhou stranu při dostatečné granularitě stránkami podporovaných rozlišení se detekované rozměry mohou od těch skutečných lišit jen o drobnou odchylku (kterou navíc může v případě potřeby uživatel sám upravit změnou parametrů). Negativní dopad na uživatelský komfort tak je tímto způsobem omezený.

4.3.3 Obrana proti otisku orientace okna prohlížeče pomocí CSS

Ve funkci `replace_orientation` je implementovaná obrana proti fingerprintingu orientace okna prohlížeče, který je popsaný v sekci 1.7.3. Obrana poskytuje tři režimy, mezi kterými může uživatel volit pomocí parametrů:

- Pomocí režimu `landscape` budou všechny výskyty hodnot orientace v pravidle `@media` nahrazeny za `landscape`, tedy za režim na šířku. To znamená, že pokud bude okno orientované na šířku, aplikují se styly bez ohledu na to, pro jakou orientaci byly původně určeny. Naopak bude-li okno orientované na výšku, neaplikují se žádné styly se specifikovanou orientací.

- Analogickým způsobem funguje režim `portrait`, který nahradí všechny výskyty hodnot orientace režimem na výšku.

- Režim `switch` umožňuje nahrazení všech hodnot typu `portrait` za hodnoty typu `landscape` a obráceně. Bude-li tedy okno orientované na šířku, aplikují se jen ty styly, které jsou určeny pro okno orientované na výšku, a obráceně.

Nahrazení klíčových slov je implementováno jednoduchou aplikací statických filtrů. Cílem obrany je zajistit, že server dostane prostřednictvím CSS nesprávnou informaci o orientaci okna nebo případně nedostane informaci žádnou.

4.3.3.1 Diskuze nedostatků

Jak bylo uvedeno už v sekci 1.7.3, ve většině případů poskytuje otisk orientace okna velmi málo informace a lze tedy spíše využít například pro detekci nasazení antifingerprintingových obran. Je proto třeba spustit tuto obranu obezřetně a využít ji spíše jako podpůrnou metodu např. pro maskování rozlišení okna. Neuvážená aplikace může uživateli snadno způsobit kontraproduktivní výsledky.

Aplikace obrany nemusí mít na vzhled stránek žádný vliv (neboť responzivní design lze v běžných případech řešit snadno bez použití této funkcionality CSS), ovšem rovněž může využívání některých stránek naprosto znehodnotit – je možné se například setkat s webovými stránkami, které neumožňují využívání v režimu na výšku. Je proto potřeba obranu aktivovat opatrně s přihlédnutím i k tomuto faktoru.

4.3.4 Obrana proti otisku pomocí HSTS

Jak jsme ukázali v sekci 1.6, hlavičku HSTS lze využít pro vytvoření supercookie identifikující návštěvníka stránek. Navrženou obranou, která je implementovaná ve funkci `remove_hsts`, je hlavičku HSTS v náhodně zvolených případech jednoduše zahodit. Uživatel může změnou celočíselného parametru této obrany definovat, s jak velkou pravděpodobností bude hlavička zahozena, pokud je v přijaté odpovědi obsažena (pokud uživatel zvolí za parametr hodnotu n, bude pravděpodobnost zahození hlavičky $\frac{1}{n}$).

Jelikož HSTS je hlavička sloužící primárně ke zvýšení bezpečnosti uživatelů na internetu, existují v implementaci tři varianty chování této obrany, mezi nimiž může uživatel vybrat pomocí sufixu názvu:

- Při zvolení `hsts-all` se bude filtrování hlavičky se zvolenou pravděpodobností provádět na všech příchozích odpovědích.
- Při zvolení `hsts-safe` se bude filtrování provádět pouze na souborech, které nejsou textové (tzn. filtrování se neprovede například na HTML souborech a JavaScriptových souborech, kde je zvýšené riziko při úspěšném útoku MITM).
- Varianta `hsts-aggressive` přidává se zvolenou pravděpodobností všem odpovědím hlavičku `Strict-Transport-Security: max-age=0` (a pokud soubor obsahuje jinou HSTS hlavičku, odstraní ji). To způsobí, že prohlížeč odstraní navštívenou doménu ze svého seznamu domén využívajících HTTPS a příští požadavek uskuteční nešifrovaným kanálem.

4.3.4.1 Dopady nasazení obrany na chování prohlížeče

Je-li hlavička HSTS odstraněna, dochází k narušení otisku prohlížeče – očekávané chování ze strany serveru je, že bude prohlížeč při další návštěvě automaticky přistupovat prostřednictvím HTTPS. Pokud k tomu nedojde, nelze na základě této informace uživatele identifikovat. Analogicky pokud server využívá k fingerprintingu větší množství domén (jako je popsáno v sekci 1.6), může být při patřičně zvolených parametrech narušeno očekávané chování prohlížeče v dostatečně velkém počtu těchto domén a uživatel tak může přijít o svou identifikovatelnost. Lepších výsledků bude obrana dosahovat v případě že vytvořená supercookie obsahuje velké množství kladných bitů. Obrana totiž dokáže změnit otisk pouze odstraněním hlavičky HSTS (tj. změnou bitu z hodnoty 1 na hodnotu 0), ale nikdy ne jejím přidáním.

Nevýhodou, která plyne přímo z podstaty této obrany, je skutečnost, že její aplikací se uživatel do určité míry vystavuje nebezpečí útoku typu MITM – pokud bude zahozena legitimně použitá hlavička HSTS na nově navštívěné doméně, bude příští spojení opět navazováno pomocí nešifrovaného protokolu, čehož může případný útočník zneužít. Nebezpečí tohoto útoku lze snížit použitím varianty `hsts-safe`, nicméně i přesto je značné a je potřeba to mít při spouštění této obrany na paměti.

Poznamenejme ještě, že v případě variant `hsts-all` a `hsts-safe` hrozí toto nebezpečí pouze u nově navštívených domén – pokud byl server s doménou využívající hlavičku HSTS navštíven již v minulosti, odstranění této hlavičky z odpovědi nebude mít na chování prohlížeče vliv. Naproti tomu varianta `hsts-aggressive` může způsobit bezpečnostní riziko u všech domén, ale zato poskytuje silnější obranu proti fingerprintingu – otisk je totiž narušen při každé návštěvě serveru, ne pouze při návštěvě první.

4.3.5 Obrana proti otisku pomocí ETag a Last-Modified

V sekci 1.3 byly rozebrány možnosti využití hlaviček `ETag` a `Last-Modified` k vytvoření supercookies. V sekci 3.1.1 byl nastíněn způsob, jakým Privoxy chrání uživatele proti sledování pomocí filtrování hlavičky `Last-Modified`. Tato obrana ovšem nijak nepůsobí v případě, že je hlavička (po syntaktické stránce) správně zformovaná. Navíc v Privoxy neexistuje obrana proti supercookies skrze hlavičku `ETag`, která představuje ještě o něco rozsáhlejší problém, neboť její hodnota nemá pevně stanovený formát, a lze proto využít jako ekvivalent standardních cookies.

Navrženou obranou, která je implementovaná funkcí `remove_etag`, je opět tyto hlavičky v náhodně vybraných případech vypouštět. S ohledem na co největší zachování funkcionality těchto hlaviček je však pravděpodobnost zahození upravena podle velikosti příchozího obsahu. Uživatel volí při spouštění akce obrany `etag` dva parametry L a H, které udávají prahové velikosti obsahu. Algoritmus pak pracuje následujícím způsobem:

1. Vzhledem k tomu, že k filtrování hlaviček a jejich následnému odeslání klientovi dochází v Privoxy ještě před zpracováním obsahu, nelze bez hlavičky `Content-Length` určit velikost obsahu. Proto není-li tato hlavička nastavena, algoritmus končí.

2. Je-li v příchozí odpovědi hlavička `Content-Length`, pak uložíme velikost, kterou udává, do proměnné S.

3. Pokud platí, že $S \leq L$, potom budou zahozeny všechny hlavičky `ETag` a `Last-Modified`.

4. Pokud platí, že $S \geq H$, potom nebude zahozena žádná hlavička.

5. Pokud platí, že $L < S < H$, pak budou zahozeny všechny hlavičky `ETag` a `Last-Modified` s pravděpodobností p, pro kterou platí vztah:

$$p = 1 - \frac{S - L}{H - L}$$

Uživatel tedy může pomocí parametrů L a H nastavit, které soubory jsou dostatečně malé, aby bylo únosné nevyužívat cache, a naopak které soubory jsou příliš velké na to, aby pro ně byla funkcionalita cache jakkoli omezena. Pro všechny ostatní bude zahození hlavičky provedeno s pravděpodobností úměrnou velikosti obsahu.

4.3.5.1 Dopady nasazení obrany na chování prohlížeče

Obrana pochopitelně negativně ovlivní fungování cache. Uživatel však má míru tohoto dopadu zcela pod kontrolou prostřednictvím parametrů – těmi však zároveň ovlivňuje i účinnost obrany. Pro správné fungování obrany je podstatné,

aby se hlavičky `ETag` a `Last-Modified` nefiltrovaly zvlášť. Pokud by došlo k filtrování pouze jedné hlavičky, mohl by server uložit supercookie do hodnoty druhé hlavičky, a obrana by tak byla neúčinná. Proto algoritmus vždy filtruje všechny hlavičky, nebo nefiltruje žádnou.

Kapitola 5

Testování a vyhodnocení výsledků

Testování výstupů této práce probíhalo na operačním systému Microsoft Windows 10 na stroji s procesorem Intel i5-4300U a 12GB operační pamětí. Použitým prohlížečem byla Mozilla Firefox verze 75.0. Pro filtrování obsahu nad stránkami využívajícím protokol HTTPS bylo využito nástroje ProxHTTPSProxyMII [129]. Kompilace zdrojových kódů proběhla rovněž na systému Microsoft Windows 10 s využitím nástroje Cygwin, a to způsobem popsaným v návodu Privoxy [130].

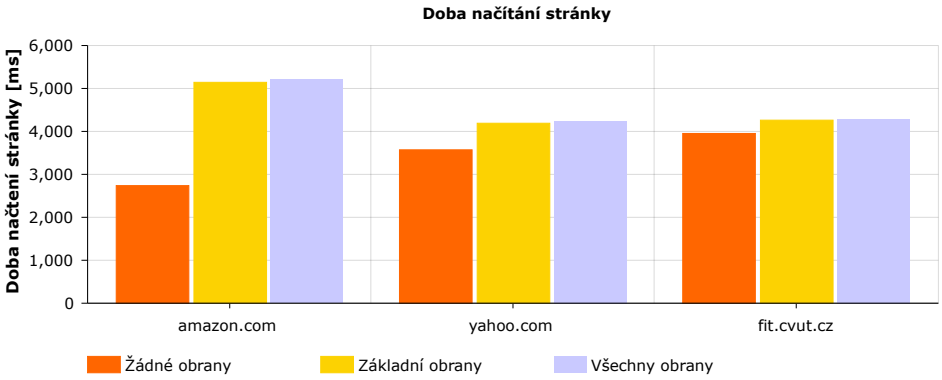
5.1 Dopad obran na rychlost načítání stránek

Při testování dopadu obran na rychlost jsme měřili dobu potřebnou pro načtení stránek `amazon.com`, `yahoo.com` a `fit.cvut.cz`. Čas byl měřen nejprve bez použití Privoxy, poté s použitím Privoxy s výchozím nastavením a zavedením obran proti fingerprintingu využívajícím canvas a objekt navigator a nakonec s nasazením všech sedmi obran proti fingerprintingu (tj. včetně pěti vytvořených v této práci). Časy zanesené v grafu 5.1 jsou vždy průměrem dvaceti měření.

Z grafu je patrné, že zanesení nových pěti obran je zdrojem pouze zanedbatelné režie. Dopad nasazení těchto obran je signifikantně menší než dopad samotného spuštění proxy serveru.

5.2 Testování účinnosti nasazených obran

Pro testování účinnosti nebylo možné použít nástroje AmIUnique a Panopticlick, protože tyto nevyužívají fingerprintingové techniky, proti nimž byla v rámci práce implementovaná obrana. Zvolili jsme proto méně používané ná-



Obrázek 5.1: Čas naměřený po načtení vybraných webových stránek při vypnutých a zapnutých antifingerprintingových obranách.

stroje, které nevyhodnocují míru identifikovatelnosti, a využili jsme je k porovnání otisku před a po zavedení obrany.

5.2.1 Testování obrany proti otisku rozlišení okna

Pro otestování obrany proti otisku rozlišení byl využit nástroj *Browserleaks* [124]. V jeho sekci dostupné z URL `https://browserleaks.com/css` lze sledovat rozlišení obrazovky reprezentující otisk prohlížeče a naměřené pomocí kaskádových stylů. Stránka v reálném čase reaguje na změnu rozlišení a to s přesností na jeden pixel.

Měření rozlišení probíhá pomocí velkého množství CSS pravidel, kde každé je pomocí vlastnosti `min-width` nebo `device-min-width` přiřazeno k některé šířce (resp. výšce) okna. Narozdíl od scénáře skutečného fingerprintingu v tomto testovacím nástroji neprobíhá informování serveru o rozměrech (vzhledem k tomu, že se jedná pouze o testovací nástroj, by to bylo zbytečné čerpání zdrojů u klienta i na serveru) a naměřené rozlišení je tedy zobrazováno pouze uživateli.

Při testování byly zvoleny parametry popsané v sekci 4.3.2 na hodnoty 100 pro výšku a 200 pro šířku a pozorováním proběhla kontrola, že při změnách rozměrů prohlížeče udává nástroj Browserleaks pouze odpovídající násobky těchto zvolených hodnot (tj. byly současně sledovány rozměry okna prohlížeče a výstupy nástroje Browserleaks).

5.2.2 Testování obrany proti otisku orientace okna

Pro testování účinnosti obrany proti fingerprintingu orientace okna bylo použita webová stránka demonstrující funkcionalitu orientace okna prohlížeče v CSS [131]. Tato stránka byla vytvořena jako názorné představení vlastnosti

orientace okna v CSS pro účely článku [132]. Přestože tento nástroj nevytváří otisk uživatele, postačí pro otestování správného chodu navržené obrany, protože implementuje funkcionalitu, která je pro vytvoření otisku využívaná.

Při testování byly postupně nastaveny všechny tři možné hodnoty parametru obrany (tj. `landscape`, `portrait` a `switch`) a vizuální kontrolou bylo ověřeno následující:

- Je-li hodnota parametru `landscape`, jsou aplikovány všechny styly při orientaci okna na šířku a žádné styly při orientaci okna na výšku.
- Je-li hodnota parametru `portrait`, jsou aplikovány všechny styly při orientaci okna na výšku a žádné styly při orientaci okna na šířku.
- Je-li hodnota parametru `switch`, nástroj udává vždy opačnou orientaci, než je skutečná orientace okna.

5.2.3 Testování obrany proti supercookie v hlavičce ETag

Nástroj *Evercookie Demo* [133] aplikuje kromě jiného i techniku vytvoření supercookie pomocí hlavičky `ETag`. Při první návštěvě uživatele se odešle požadavek na server a v odpovědi přijme klient i ETag, který pro něj server vytvoří. Pokud se při další návštěvě prokáže uživatel nějakou hodnotou hlavičky, server uživatele rozpozná. Stránka potom zobrazuje hodnotu čítače, který měří počet návštěv uživatele na základě této techniky.

Jako druhotný zdroj informace využívá nástroj běžnou cookie, tzn. restartování hodnoty čítače lze dosáhnout pouze, pokud před návštěvou současně vymažeme cache a zahodíme cookies. Pro účely našeho experimentu jsme proto cookies nechali vymazat před každým obnovením stránky, takže byla hodnota čítače závislá pouze na hlavičce `ETag`.

Při běžném chování prohlížeče (tj. bez nasazení obrany) se čítač při každé návštěvě inkrementuje. Testování správné funkčnosti obrany proběhlo ověřením následujících jednoduchých situací:

- Je-li hodnota parametru L větší než velikost stránky, ke které je hlavička `ETag` připojena (tj. přibližně 3317 B), bude hodnota čítače při každé návštěvě rovna nule (protože hlavička `ETag` byla vždy zahozena).
- Je-li hodnota parametru H menší než velikost stránky, bude hodnota čítače při každé návštěvě inkrementována (tj. aplikace obrany nijak neovlivní standardní chod stránek).

Pro testování očekávaného poměru zahozených hlaviček ku ponechaným jsme zvolili následující experiment: Parametry L a H byly nastaveny na hodnoty 1000 a 5635, takže při velikosti souboru 3317 B by hlavička měla být

Návštěva	1.	2.	3.	4.	5.	6.	7.	8.
1.	0	456	454	466	457	459	455	469
2.	456	0	458	474	459	461	443	467
3.	454	458	0	286	455	463	463	455
4.	466	474	286	0	465	471	467	463
5.	457	459	455	465	0	294	466	468
6.	459	461	463	471	294	0	452	458
7.	455	443	463	467	466	452	0	456
8.	469	467	455	463	468	458	456	0

Tabulka 5.1: Počet rozdílných fontů naměřených mezi jednotlivými návštěvami nástroje Browserize.

zahozena přibližně v polovině případů. Po provedení 100 požadavků byla detekována nenulová hodnota čítače v 49 případech, což indikuje, že hlavička byla zahozena v 51 případech.

Podobně pro hodnoty parametrů 1000 a 10269 byla hlavička zahozena v 72 ze 100 případů (oproti očekáváným 75) a pro hodnoty 0 a 4974 byla zahozena ve 36 ze 100 případů (oproti očekávaným 33). Naměřené hodnoty jsou tedy uspokojivě blízko očekávaným.

5.2.4 Testování obrany proti otisku fontů

Nástroj Browserize [125] vytváří také otisk fontů pomocí CSS. Testuje při tom přítomnost 1428 fontů na zařízení uživatele. Po dokončení tvorby otisku zobrazí uživateli samotný otisk (hash vytvořený z naměřeného seznamu) ale rovněž i výpis všech fontů, které byly na zařízení detekovány. Bez nasazení obrany je seznam fontů neměnný napříč jednotlivými návštěvami.

Pro testování byl uložen výpis detekovaných fontů v osmi návštěvách (při aplikované obraně) a byly porovnány rozdíly v těchto výpisech mezi jednotlivými návštěvami. V tabulce 5.1 poskytujeme informace o počtu rozdílně naměřených fontů mezi návštěvami. Z tabulky vyplývá, že se otisk napříč návštěvami lišil nejméně o 286 položek. Bylo by tedy velmi obtížné z pohledu serveru jednotlivé požadavky spárovat.

5.2.5 Testování obrany proti HSTS supercookie

Pro testování obrany proti vytvoření HSTS supercookie byl využit nástroj *HSTS super-cookie PoC* [123]. V běžném provozu tento nástroj při první návštěvě vytvoří pomocí hlavičky HSTS uživateli supercookie a oznámí mu to

Parametr	Různých ID	Nových ID	Spárovaných ID	Cizích ID
2	46	27	4	19
5	38	14	12	24
10	21	5	29	16
20	14	4	36	10
40	12	3	38	9

Tabulka 5.2: Výsledky měření při padesáti požadavcích s různým vstupním parametrem obrany HSTS.

zprávou „*Your cookie is set to...*“ s příslušnou hodnotou šestnáctibitového identifikátoru. Při příštích návštěvách je stejný identifikátor zobrazen zprávou „*Your cookie is...*“, což signalizuje, že byl uživatel rozpoznán.

Po nasazení obrany `hsts-aggressive` s parametrem 5 (tj. k úpravě hlaviček dojde v $\frac{1}{5}$ případů) jsme provedli 50 požadavků. V odpovědi na tyto požadavky jsme získali následující výstupy:

- Nástroj vykázal celkem 38 různých identifikátorů.
- 14 identifikátorů bylo nově vytvořeno (indikováno zprávou „*Your cookie is set to...*“).
- 12 požadavků bylo spárováno s některým předchozím požadavkem (odpovědi serveru vykázaly stejný identifikátor).
- Nástroj vykázal pro 24 požadavků identifikátor, který byl mezi dosavadními požadavky unikátní a zároveň mylně rozpoznal náš požadavek jako požadavek jiného uživatele (indikováno zprávou „*Your cookie is...*“ v případě, kdy jsme stejný identifikátor v předchozích odpovědích nezískali).

Analogicky jsme měření provedli pro hodnoty parametru 2, 10, 20 a 40. Výsledky jsou zaneseny v tabulce 5.2. Jak jsme očekávali, se snižující se pravděpodobností zahození hlavičky dochází k větší frekvenci spárování požadavků na základě identifikátoru a obrana se tedy stává méně efektivní.

Závěr

V práci bylo pojednáno o fingerprintingových technikách, které v posledních letech prošly podstatným vývojem nebo jsou méně používané. Rozebrali jsme, jakým způsobem se v současnosti lze bránit proti fingerprintingu a k jakým opatřením ve snaze fingerprinting omezovat přistupují současné verze webových prohlížečů. Představili jsme program Privoxy a obrany, které proti fingerprintingu nasazuje. Shrnuli jsme rovněž předchozí práci na rozšíření těchto antifingerprintingových obran.

Navrhli jsme možné obrany proti pěti vybraným fingerprintingovým technikám, pro jejichž implementaci poskytuje Privoxy vhodné podmínky. Vysvětlili jsme způsob fungování a implementaci těchto obran a nastínili jsme dopady, které může mít jejich nasazení na chování navštívených webových stránek. Fungování obran jsme testovali pomocí představených dostupných nástrojů a experimentálně jsme měřili dopady nasazení těchto obran na rychlost načítání vybraných webových stránek.

Obrany navržené a implementované v této práci je potřeba používat s rozvahou, aby nepřinesly uživateli kontraproduktivní výsledky. Prostor pro další rozšíření práce je značný – jak bylo v textu demonstrováno, fingerprintingových technik je nepřeberné množství a obrana, kterou v současnosti poskytují webové prohlížeče, je stále omezená. V navazující práci by proto bylo možné věnovat pozornost dalším fingerprintingovým technikám, případně rozšířit nebo upravit stávající obrany podle měnících se trendů ve vytváření digitálních otisků prohlížeče.

Literatura

[1] Confessore, N.: Cambridge Analytica and Facebook: The Scandal and the Fallout So Far. *The New York Times [online]*, 2018, [cit. 2020-03-02]. Dostupné z: https://www.nytimes.com/2018/04/04/us/politics/cambridge-analytica-scandal-fallout.html

[2] Penland, J.: Browser Cookies: What Are They & Why Should You Care? *Who Is Hosting This? [online]*, 2020, [cit. 2020-03-02]. Dostupné z: https://www.whoishostingthis.com/resources/cookies-guide/

[3] Publications Office of the European Union: Směrnice Evropského parlamentu a Rady 2002/58/ES ze dne 12. července 2002 o zpracování osobních údajů a ochraně soukromí v odvětví elektronických komunikací (Směrnice o soukromí a elektronických komunikacích). *Úřední věstník [online]*, 2002, [cit. 2020-03-03]. Dostupné z: https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002L0058:CS:HTML

[4] Briz, N.: This is Your Digital Fingerprint. *The Mozilla Blog [online]*, 2018, [cit. 2020-03-02]. Dostupné z: https://blog.mozilla.org/internetcitizen/2018/07/26/this-is-your-digital-fingerprint/

[5] Privoxy Developers: Privoxy - Home Page. [online], 2020, [cit. 2020-03-03]. Dostupné z: https://www.privoxy.org/

[6] Prášil, A.: Website user tracking. 2016, diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií.

[7] Sixta, J.: Obrana proti fingerprintingu do Privoxy. 2018, diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií.

[8] We Are Social: Digital in 2019. [online], 2020, [cit. 2020-03-04]. Dostupné z: https://wearesocial.com/global-digital-report-2019

[9] AdSense Help: Personalized and non-personalized ads. [online], 2020, [cit. 2020-03-05]. Dostupné z: https://support.google.com/adsense/answer/9007336?hl=en

[10] Szymielewicz K. a Budington, B.: Personalized and non-personalized ads. [online], 2018, [cit. 2020-03-05]. Dostupné z: https://www.eff.org/deeplinks/2018/06/gdpr-and-browser-fingerprinting-how-it-changes-game-sneakiest-web-trackers

[11] Laperdrix, P.: Browser Fingerprinting: An Introduction and the Challenges Ahead. [online], 2019, [cit. 2020-03-05]. Dostupné z: https://blog.torproject.org/browser-fingerprinting-introduction-and-challenges-ahead

[12] Ashouri, M. a Asadian, H.: Large-Scale Analysis of Sophisticated Web Browser Fingerprinting Scripts. *The Fourteenth International Conference on Internet Monitoring and Protection*, 2018, [cit. 2020-03-08]. Dostupné z: https://hal.archives-ouvertes.fr/hal-01811691/document

[13] Piano Media: Piano – The Digital Business Platform. [online], 2020, [cit. 2020-03-08]. Dostupné z: https://piano.io/

[14] Unger, T. a spol.: SHPF: Enhancing HTTP(S) Session Security with Browser Fingerprinting (extended preprint). *2013 International Conference on Availability, Reliability and Security*, 2013, [cit. 2020-03-21]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.372.8339&rep=rep1&type=pdf

[15] Adobe Corporate Communications: Flash & The Future of Interactive Content. [online], 2017, [cit. 2020-03-04]. Dostupné z: https://theblog.adobe.com/adobe-flash-update/

[16] The Chromium Projects: Flash roadmap - shipped Changes. [online], 2019, [cit. 2020-03-04]. Dostupné z: https://www.chromium.org/flash-roadmap#TOC-Summary

[17] Mozilla: Bug 1519434 - Remove "Always Activate"and "Remember this decision"Flash options in Firefox 69. [online], 2019, [cit. 2020-03-04]. Dostupné z: https://bugzilla.mozilla.org/show_bug.cgi?id=1519434

[18] Oliver, J.: Fingerprinting the Mobile Web. 2018, individual Project Final Report, Imperial College of Science, Technology and Medicine.

[19] MDN contributors: Battery Status API. [online], 2020, [cit. 2020-03-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Battery_Status_API

[20] Can I use...: Support tables for HTML5, CSS3, etc – Battery Status API. [online], 2019, [cit. 2020-03-04]. Dostupné z: https://caniuse.com/#feat=battery-status

[21] Olejnik, L. a spol.: The leaking battery. *Data Privacy Management, and Security Assurance*, 2015, [cit. 2020-03-04]. Dostupné z: https://eprint.iacr.org/2015/616.pdf

[22] W3C: Battery Status API – W3C Candidate Recommendation 08 May 2012. [online], 2012, [cit. 2020-03-05]. Dostupné z: https://www.w3.org/TR/2012/CR-battery-status-20120508/

[23] Firefox Site Compatibility: Battery Status API has been removed. [online], 2016, [cit. 2020-03-04]. Dostupné z: https://www.fxsitecompat.dev/en-CA/docs/2016/battery-status-api-has-been-removed/

[24] W3C: Gamepad – W3C Editor's Draft 25 February 2020. [online], 2020, [cit. 2020-03-12]. Dostupné z: https://w3c.github.io/gamepad/#widl-Gamepad-id

[25] MDN contributors: Using the Gamepad API. [online], 2019, [cit. 2020-03-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Gamepad_API/Using_the_Gamepad_API

[26] Graham, S.: Chromium-discuss › Gamepad. [online], 2019, [cit. 2020-03-04]. Dostupné z: https://groups.google.com/a/chromium.org/forum/#\protect\relax\kern-.16667emtopic/chromium-discuss/ud5jwbaJUc4

[27] Wichary, M.: Jumping the Hurdles with the Gamepad API. [online], 2012, [cit. 2020-03-05]. Dostupné z: https://www.html5rocks.com/en/tutorials/doodles/gamepad/

[28] MDN contributors: WebRTC API. [online], 2019, [cit. 2020-03-05]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API

[29] BrowserLeaks: WebRTC Leak Test. [online], 2020, [cit. 2020-03-05]. Dostupné z: https://browserleaks.com/webrtc

[30] Dvořák, J.: Ochrana anonymity uživatele v internetu. 2018, bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií.

[31] Article19: Privacy and consent in the age of browsers: The question of WebRTC. [online], 2018, [cit. 2020-03-05]. Dostupné z: https://www.article19.org/resources/privacy-and-consent-in-the-age-of-browsers-the-question-of-webrtc/

[32] MDN Contributors: DNT. [online], 2020, [cit. 2020-03-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Touch_events

[33] Schepers, D. a spol.: Touch Events - Level 2. [online], 2019, [cit. 2020-03-16]. Dostupné z: https://w3c.github.io/touch-events/

[34] Perry, M. a spol.: The Design and Implementation of the Tor Browser [DRAFT]. [online], 2018, [cit. 2020-03-11]. Dostupné z: https://2019.www.torproject.org/projects/torbrowser/design/

[35] MDN Contributors: ETag. [online], 2019, [cit. 2020-03-12]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/ETag

[36] Janc, A. a Zalewski, M.: Technical analysis of client identification mechanisms. [online], 2020, [cit. 2020-03-12]. Dostupné z: https://www.chromium.org/Home/chromium-security/client-identification-mechanisms

[37] MDN Contributors: ETag. [online], 2019, [cit. 2020-03-13]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Encoding

[38] Calvano, P.: Compression. [online], 2019, [cit. 2020-03-13]. Dostupné z: https://almanac.httparchive.org/en/2019/compression

[39] Butler, J. a spol.: A Proposal for Shared Dictionary Compression over HTTP. [online], 2008, [cit. 2020-03-13]. Dostupné z: https://lists.w3.org/Archives/Public/ietf-http-wg/2008JulSep/att-0441/Shared_Dictionary_Compression_over_HTTP.pdf

[40] mef@chromium.org a spol.: Issue 327783: SDCH support may be used to track user. [online], 2013, [cit. 2020-03-13]. Dostupné z: https://bugs.chromium.org/p/chromium/issues/detail?id=327783

[41] Sleevi, R. a spol.: Intent to Unship: SDCH. [online], 2016, [cit. 2020-03-13]. Dostupné z: https://groups.google.com/a/chromium.org/forum/#\protect\relax\kern-.16667emtopic/blink-dev/nQl0ORHy7sw

[42] MDN contributors: HTTP Public Key Pinning (HPKP). [online], 2020, [cit. 2020-03-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Public_Key_Pinning

[43] Russell, A.: What Is an X.509 Certificate? [online], 2019, [cit. 2020-03-07]. Dostupné z: https://www.ssl.com/faqs/what-is-an-x-509-certificate/

[44] Amann, J. a spol.: Mission accomplished?: HTTPS security after diginotar. *Proceedings of the 2017 Internet Measurement Conference*, 2017, [cit. 2020-03-07]. Dostupné z: https://ralphholz.science/publications/MissionAccomplishedHttpsSecurityAfterDiginotar.pdf

[45] Cooper, D. a spol.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. [online], 2008, [cit. 2020-03-07]. Dostupné z: https://www.ietf.org/rfc/rfc5280.txt

[46] Helme, S.: HPKP is no more! [online], 2020, [cit. 2020-03-07]. Dostupné z: https://scotthelme.co.uk/hpkp-is-no-more/

[47] Amazon Web Services: Alexa Top Sites. [online], 2020, [cit. 2020-03-08]. Dostupné z: https://aws.amazon.com/marketplace/pp/B07QK2XWNV

[48] Sallai, T.: Why certificate pinning with HPKP is a bad idea. [online], 2018, [cit. 2020-03-07]. Dostupné z: https://advancedweb.hu/why-certificate-pinning-with-hpkp-is-a-bad-idea/

[49] Helme, S.: Using security features to do bad things. [online], 2016, [cit. 2020-03-07]. Dostupné z: https://scotthelme.co.uk/using-security-features-to-do-bad-things/

[50] Chrome Platform Status: Remove HTTP-Based Public Key Pinning (removed). [online], 2018, [cit. 2020-03-07]. Dostupné z: https://www.chromestatus.com/feature/5903385005916160

[51] Firefox Site Compatibility: HTTP Public Key Pinning is no longer supported. [online], 2019, [cit. 2020-03-07]. Dostupné z: https://www.fxsitecompat.dev/en-CA/docs/2019/http-public-key-pinning-is-no-longer-supported/

[52] Keeler, D.: bug 1412438 - add preference to disable HPKP by default r=jcj. [online], 2019, [cit. 2020-03-07]. Dostupné z: https://hg.mozilla.org/mozilla-central/rev/d791bfa31f08

[53] MDN Contributors: Strict-Transport-Security. [online], 2020, [cit. 2020-03-08]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security

[54] Marlinspike, M.: Software » sslstrip. [online], 2012, [cit. 2020-03-08]. Dostupné z: https://moxie.org/software/sslstrip/

[55] Hodges, J. a spol.: HTTP Strict Transport Security (HSTS). [online], 2012, [cit. 2020-03-08]. Dostupné z: https://www.rfc-editor.org/rfc/rfc6797.txt

[56] mmenke@chromium.org a spol.: Issue 774643: Clearing non-incognito data results in retainining history in incognito's TransportSecurityPersister. [online], 2017, [cit. 2020-03-13]. Dostupné z: https://bugs.chromium.org/p/chromium/issues/detail?id=774643

[57] Can I use...: Support tables for HTML5, CSS3, etc – Strict Transport Security. [online], 2020, [cit. 2020-03-08]. Dostupné z: https://caniuse.com/#feat=stricttransportsecurity

[58] W3Techs: Usage statistics of HTTP Strict Transport Security for websites. [online], 2020, [cit. 2020-03-08]. Dostupné z: https://w3techs.com/technologies/details/ce-hsts

[59] Takei, N. a spol.: Web Browser Fingerprinting using only Cascading Style Sheets. *10th International Conference on Broadband and Wireless Computing, Communication and Applications*, 2015, [cit. 2020-03-09]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7424801

[60] Robinson, S.: flipping typical: preview text & compare your fonts easily. [online], 2009, [cit. 2020-03-11]. Dostupné z: http://flippingtypical.com/

[61] MDN Contributors: font-family. [online], 2019, [cit. 2020-03-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/font-family

[62] Böhmer, J.: CrookedStyleSheets. [online], 2018, [cit. 2020-03-11]. Dostupné z: https://github.com/jbtronics/CrookedStyleSheets

[63] MDN Contributors: font-family. [online], 2019, [cit. 2020-03-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/%40font-face

[64] Electronic Frontier Foundation: Panopticlick. [online], 2015, [cit. 2020-03-11]. Dostupné z: https://panopticlick.eff.org/

[65] Laperdrix, P. a spol.: AmIUnique. [online], 2016, [cit. 2020-03-11]. Dostupné z: https://amiunique.org/

[66] MDN Contributors: @media. [online], 2020, [cit. 2020-03-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/%40media

[67] Roberts, A.: Vendor-specific Properties. [online], 2020, [cit. 2020-03-11]. Dostupné z: https://www.sitepoint.com/vendor-specific-properties/

[68] MDN Contributors: Vendor Prefix. [online], 2020, [cit. 2020-03-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Vendor_Prefix

[69] MDN Contributors: @supports. [online], 2020, [cit. 2020-03-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/%40supports

[70] MDN Contributors: @keyframes. [online], 2020, [cit. 2020-03-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/%40keyframes

[71] MDN contributors: HTTP authentication. [online], 2019, [cit. 2020-03-13]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication

[72] James, P.: HTTP Authentication with HTML Forms. [online], 2006, [cit. 2020-03-13]. Dostupné z: https://www.peej.co.uk/articles/http-auth-with-html-forms.html

[73] Grossman, J.: Tracking users with Basic Auth. [online], 2007, [cit. 2020-03-13]. Dostupné z: https://blog.jeremiahgrossman.com/2007/04/tracking-users-without-cookies.html

[74] Jorgensen, Z. a Yu, T.: On Mouse Dynamics as a Behavioral Biometric for Authentication. *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, 2011, [cit. 2020-03-14]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.310.4320&rep=rep1&type=pdf

[75] Deutschmann, I. a spol.: Continuous Authentication Using Behavioral Biometrics. *IT Professional*, 2013, [cit. 2020-03-14]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6544522

[76] Sharma, S.: What is the Use of Accelerometer in Mobile Devices? (Updated). [online], 2019, [cit. 2020-03-15]. Dostupné z: https://www.credencys.com/blog/accelerometer/

[77] Devices and Sensors Working Group: DeviceOrientation Event Specification. [online], 2019, [cit. 2020-03-15]. Dostupné z: https://w3c.github.io/deviceorientation/

[78] Aviv, A. a spol.: Practicality of Accelerometer Side Channels on Smartphones. *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, [cit. 2020-03-15]. Dostupné z: https://www.cs.swarthmore.edu/~aviv/papers/aviv-acsac12-accel.pdf

[79] Cai, L. a Chen, H.: TouchLogger: Inferring Keystrokes On Touch Screen From Smartphone Motion. *Proceedings of the 6th USENIX conference on Hot topics in security*, 2011, [cit. 2020-03-15]. Dostupné z: https://static.usenix.org/events/hotsec11/tech/final_files/Cai.pdf

[80] Xu, Z. a spol.: TapLogger: inferring user inputs on smartphone touchscreens using on-board motion sensors. *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Network*, 2012, [cit. 2020-03-15]. Dostupné z: http://www.cse.psu.edu/~sxz16/papers/taplogger.pdf

[81] Luca, A. a spol.: Touch me once and I know it's you! Implicit Authentication based on Touch Screen Patterns. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012, [cit. 2020-03-17]. Dostupné z: http://141.84.8.93/pubdb/publications/pub/deluca2012chi/deluca2012chi.pdf

[82] Saevanee, H. a Bhatarakosol, P.: User Authentication using Combination of Behavioral Biometrics over the Touchpad acting like Touch screen of Mobile Device. *2008 International Conference on Computer and Electrical Engineering*, 2008, [cit. 2020-03-17]. Dostupné z: https://www.it.iitb.ac.in/frg/wiki/images/2/2b/113050033_Paper8.pdf

[83] MDN Contributors: DNT. [online], 2019, [cit. 2020-03-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/DNT

[84] MDN Contributors: Cross-Origin Resource Sharing (CORS). [online], 2020, [cit. 2020-03-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[85] Bortz, A. a spol.: Exposing Private Information by Timing Web Applications. *Proceedings of the 16th international conference on World Wide Web, [online]*, 2007, [cit. 2020-03-14]. Dostupné z: https://www.abortz.net/papers/timingweb.pdf

[86] Felten, E. a Chneider, M.: Timing Attacks on Web Privacy. *7th ACM Conference on Computer Communications Security*, 2000, [cit. 2020-03-14]. Dostupné z: https://sip.cs.princeton.edu/pub/webtiming.pdf

[87] MDN Contributors: navigator.hardwareConcurrency. [online], 2019, [cit. 2020-03-18]. Dostupné z: https://developer.mozilla.org/

en-US/docs/Web/API/NavigatorConcurrentHardware/hardwareConcurrency

[88] MDN Contributors: Web Workers API. [online], 2019, [cit. 2020-03-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API

[89] OFTN – Open Source Working Group: Core Estimator. [online], 2017, [cit. 2020-03-18]. Dostupné z: https://github.com/oftn-oswg/core-estimator

[90] Mowery, K. a spol.: Fingerprinting Information in JavaScript Implementations. *Proceedings of W2SP*, 2011, [cit. 2020-03-17]. Dostupné z: https://hovav.net/ucsd/dist/jspriv.pdf

[91] Perry, M.: Do Not Beg: Moving Beyond DNT through Privacy by Design. [online], 2012, [cit. 2020-03-18]. Dostupné z: https://www.w3.org/2012/dnt-ws/position-papers/21.pdf

[92] Electronic Frontier Foundation: Privacy Badger. [online], 2020, [cit. 2020-03-18]. Dostupné z: https://www.eff.org/privacybadger

[93] Laperdrix, P. a spol.: Browser Fingerprinting: A survey. 2019, [cit. 2020-03-18]. Dostupné z: https://arxiv.org/pdf/1905.01051.pdf

[94] Eckersley, P.: How Unique Is Your Web Browser? *International Symposium on Privacy Enhancing Technologies Symposium*, 2010, [cit. 2020-03-18]. Dostupné z: http://asturias.axtur.com/wp-content/uploads/2014/01/browser-uniqueness.pdf

[95] Al-Fannah, N. M. a spol.: Too Little Too Late: Can We Control Browser Fingerprinting? *Journal of Intellectual Capital*, 2020, [cit. 2020-03-18]. Dostupné z: http://www.chrismitchell.net/Papers/tltlcw.pdf

[96] Doty, N.: Mitigating Browser Fingerprinting in Web Specifications. [online], 2019, [cit. 2020-03-22]. Dostupné z: https://www.w3.org/TR/fingerprinting-guidance/

[97] Snyder, P. a spol.: Browser Feature Usage on the Modern Web. *Proceedings of the 2016 Internet Measurement Conference*, 2016, [cit. 2020-03-22]. Dostupné z: https://arxiv.org/pdf/1605.06467.pdf

[98] Das, A. a spol.: Every Move You Make: Exploring Practical Issues in Smartphone Motion Sensor Fingerprinting and Countermeasures. *Proceedings on Privacy Enhancing Technologies*, 2018, [cit. 2020-03-22]. Dostupné z: https://www.degruyter.com/downloadpdf/j/popets.2018.2018.issue-1/popets-2018-0005/popets-2018-0005.pdf

[99] The Tor Project: Tor Project | History. [online], 2020, [cit. 2020-03-24]. Dostupné z: https://www.torproject.org/about/history/

[100] Tseng, E. a Barnes, R.: Tor at the Heart: Firefox. [online], 2016, [cit. 2020-03-24]. Dostupné z: https://blog.torproject.org/tor-heart-firefox

[101] Englehardt, S.: Firefox 72 blocks third-party fingerprinting resources. [online], 2020, [cit. 2020-03-24]. Dostupné z: https://blog.mozilla.org/security/2020/01/07/firefox-72-fingerprinting/

[102] Disconnect: Canonical repository for the Disconnect services file. [online], 2020, [cit. 2020-03-24]. Dostupné z: https://github.com/disconnectme/disconnect-tracking-protection

[103] Disconnect: Tracker Descriptions. [online], 2020, [cit. 2020-03-24]. Dostupné z: https://github.com/disconnectme/disconnect-tracking-protection/blob/master/descriptions.md

[104] Perry, M. a spol.: Prompt (w/ Site Permission) before allowing content to extract canvas data (Tor 6253). [online], 2014, [cit. 2020-03-25]. Dostupné z: https://bugzilla.mozilla.org/show_bug.cgi?id=967895

[105] Edelstein, A.: Reduce precision of time exposed by Javascript (Tor 1517). [online], 2015, [cit. 2020-03-25]. Dostupné z: https://bugzilla.mozilla.org/show_bug.cgi?id=1217238

[106] Edelstein, A. a spol.: When privacy.resistFingerprinting=true, dynamically round content dimensions. [online], 2017, [cit. 2020-04-10]. Dostupné z: https://bugzilla.mozilla.org/show_bug.cgi?id=1407366

[107] Vigier, N.: New Release: Tor Browser 9.0. [online], 2019, [cit. 2020-04-04]. Dostupné z: https://blog.torproject.org/new-release-tor-browser-90

[108] The Tor Project: Users – Tor Metrics. [online], 2020, [cit. 2020-03-25]. Dostupné z: https://metrics.torproject.org/userstats-relay-country.html

[109] Mozilla: Firefox Public Data Report. [online], 2020, [cit. 2020-03-25]. Dostupné z: https://data.firefox.com/dashboard/user-activity

[110] Stat Counter: StatCounter Global Stats - Browser, OS, Search Engine including Mobile Usage Share. [online], 2020, [cit. 2020-03-21]. Dostupné z: https://gs.statcounter.com/

[111] Schuh, J.: Building a more private web. [online], 2019, [cit. 2020-03-25]. Dostupné z: https://www.blog.google/products/chrome/building-a-more-private-web

[112] Chromium Blog: Improving privacy and security on the web. [online], 2019, [cit. 2020-03-25]. Dostupné z: https://blog.chromium.org/2019/05/improving-privacy-and-security-on-web.html

[113] The Chromium Projects: The Privacy Sandbox. [online], 2020, [cit. 2020-03-25]. Dostupné z: https://www.chromium.org/Home/chromium-privacy/privacy-sandbox

[114] Chromium Blog: Building a more private web: A path towards making third party cookies obsolete. [online], 2020, [cit. 2020-03-25]. Dostupné z: https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html

[115] Chromium Blog: Potential uses for the Privacy Sandbox. [online], 2019, [cit. 2020-03-25]. Dostupné z: https://blog.chromium.org/2019/08/potential-uses-for-privacy-sandbox.html

[116] Harisson, C.: Aggregated Reporting API. [online], 2020, [cit. 2020-03-25]. Dostupné z: https://github.com/csharrison/aggregate-reporting-api

[117] Al-Fannah, N. M. a spol.: Beyond Cookie Monster Amnesia: Real World Persistent Online Tracking. *International Conference on Information Security*, 2018, [cit. 2020-03-21]. Dostupné z: https://arxiv.org/pdf/1905.09581.pdf

[118] Majestic: The Majestic Million. [online], 2020, [cit. 2020-03-21]. Dostupné z: https://majestic.com/reports/majestic-million

[119] Google: Analytics. [online], 2020, [cit. 2020-03-21]. Dostupné z: https://analytics.google.com/

[120] Maone, G.: NoScript - JavaScript/Java/Flash blocker for a safer Firefox experience! [online], 2020, [cit. 2020-04-10]. Dostupné z: https://noscript.net/

[121] kkapsner: CanvasBlocker: A Firefox extension to protect from being fingerprinted. [online], 2020, [cit. 2020-04-10]. Dostupné z: https://github.com/kkapsner/CanvasBlocker

[122] Gómez-Boix, A. a spol.: Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. *Proceedings of the 2018 World Wide Web Conference*, 2020, [cit. 2020-04-18]. Dostupné z: https://hal.inria.fr/hal-01718234v2/document

[123] Zhirov, A.: JavaScript-less HSTS super-cookie PoC. [online], 2019, [cit. 2020-04-19]. Dostupné z: http://hsts.nevkontakte.com/

[124] BrowserLeaks: BrowserLeaks - Web Browser Fingerprinting - Browsing Privacy. [online], 2020, [cit. 2020-04-18]. Dostupné z: https://browserleaks.com

[125] Weber, D. J.: Browserize - The Browser Characterization Library. [online], 2019, [cit. 2020-04-18]. Dostupné z: https://privacycheck.sec.lrz.de

[126] MDN Contributors: Last-Modified. [online], 2020, [cit. 2020-03-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Last-Modified

[127] Kumar, M. a spol.: Shuffle a given array using Fisher–Yates shuffle Algorithm. [online], 2020, [cit. 2020-04-04]. Dostupné z: https://www.geeksforgeeks.org/shuffle-a-given-array-using-fisher-yates-shuffle-algorithm/

[128] Daggett, J. a spol.: CSS Fonts Module Level 3. [online], 2018, [cit. 2020-04-04]. Dostupné z: https://drafts.csswg.org/css-fonts-3/#font-family-prop

[129] whenever: ProxHTTPSProxyMII: Reloaded. [online], 2018, [cit. 2020-04-19]. Dostupné z: https://prxbx.com/forums/showthread.php?tid=2172

[130] Privoxy: Privoxy 3.0.28 User Manual. [online], 2018, [cit. 2020-04-19]. Dostupné z: https://www.privoxy.org/user-manual/installation.html

[131] Firdaus, T.: CSS Orientation - Hongkiat.com. [online], 2017, [cit. 2020-04-28]. Dostupné z: https://hongkiat.github.io/css-orientation-styles/

[132] Firdaus, T.: UI Design: Applying CSS Based on Screen Orientation. [online], 2017, [cit. 2020-04-20]. Dostupné z: https://www.hongkiat.com/blog/css-orientation-styles/

[133] Gut, K.: Evercookie Demo. [online], 2019, [cit. 2020-04-28]. Dostupné z: https://cable.ayra.ch/toys/track.php

Příloha A

Seznam použitých zkratek

API Application Programming Interface

CSS Cascading Style Sheets

HPKP HTTP Public Key Pinning

HSTS HTTP Strict Transport Security

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

MITM Man in The Middle

NAT Network Address Translation

P2P Peer-to-peer

PKI Public Key Infrastructure

SSL Secure Sockets Layer

TLS Transport Layer Security

TOFU Trust on First Use

Příloha **B**

Obsah přiložené SD karty

- `readme.txt` ..... podrobnější popis obsahu SD karty
- `exe` ..... spustitelná verze se základním nastavením
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Hanak_Samuel_2020.pdf` ..... text práce ve formátu PDF