VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ZJIŠTĚNÍ IDENTITY PROHLÍŽEČE POMOCÍ WEBASSEMBLY

BROWSER FINGERPRINTING USING WEB ASSEMBLY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MATÚŠ ŠKUTA
AUTHOR

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

BRNO 2020

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

22937

Student: **Škuta Matúš**
Program: Informační technologie
Název: **Zjištění identity prohlížeče pomocí WebAssembly**
Browser Fingerprinting Using Web Assembly
Kategorie: Bezpečnost
Zadání:

1. Nastudujte jazyk WebAssembly a jeho současnou podporu v existujících prohlížečích.
2. Seznamte se s metodami a nástroji získávajícími otisk prohlížeče, např. JSTemplate, FP-Scanner, FingerprintJS, Panopticlick, AmIUnique.org.
3. Navrhněte program ve WebAssembly, který vytváří otisk prohlížeče.
4. Program implementujte a vytvořte webovou stránku pro jeho demonstraci.
5. Implementaci otestujte vůči stávajícím technikám detekce otisku prohlížeče (existující rozšíření jako JS Restrictor, Web API Manager, či specializované prohlížeče Brave, Tor Browser).
6. Zhodnoťe dopady projektu a navrhněte možná pokračování.

Literatura:

- Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. 2019. Browser Fingerprinting: A survey.
- Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. FP-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies. In Proceedings of the 27th USENIX Security Symposium. Baltimore, United States.
- Michael Schwarz, Florian Lackner, and Daniel Gruss. 2019. JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits. In Network and Distributed Systems Security (NDSS) Symposium.

Pro udělení zápočtu za první semestr je požadováno:

- První 3 body zadání včetně technické zprávy.

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

Vedoucí práce: **Polčák Libor, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 16. října 2019

Abstrakt

Hlavným cieľom tejto bakalárskej práce, je implementácia identifikácie zariadení, pomocou technológie Web Assembly. V práci si rozoberieme už existujúce metódy identifikácie zariadení, metódy na zabránenie identifikácie a spôsoby, ako tieto rôzne zábrany obísť. Tak isto sa zoznámime s prehliadačom Brave, ktorý sa snaží zredukovať identifikáciu zariadení na internete. Vysvetlíme si, ako funguje Web Assembly, aké sú jeho pozitíva, negatíva a či budeme schopný vďaka tejto novej technológii obísť rôzne obrany proti identifikácii zariadení. Ďalej sa zoznámime s niekoľkými webovými API, ktoré budeme využívať pri identifikácii zariadení a predstavíme si aj pár rozšírení, ktorých cieľom je zabrániť, alebo úplne obmedziť získanie identifikácie zariadení. Už existujúce bezpečnostné rozšírenia, obmedzujú činnosť Web Assembly, napríklad táto práca ukazuje, že rozšírenie Web API Manager dokáže neutralizovať testovaciu stránku implementovanú v rámci práce.

Abstract

The main goal of this bachelor thesis is the implementation of device identification using Web Assembly technology. In this work we are discussing the existing methods of device identification, methods to prevent identification and ways to circumvent these barriers. We are also getting acquainted with the Brave browser, which seeks to reduce the identification of devices on the Internet. We are explaining how Web Assembly works, what are its positives, negatives and if we are able to bypass various defences against device identification thanks to this new technology. Next, we are looking at the several web APIs we are using to identify devices, and we are introducing a few extensions designed to prevent or completely limit device identification. Existing security extensions limit the activity of the Web Assembly, for example this work shows that the Web API Manager extension can neutralize the test page implemented in this work.

Klíčová slova

Web Assembly, WASM, identifikácia zariadení, web, fingerprintovanie

Keywords

Web Assembly, WASM, device identification, web, fingerprinting

Citace

ŠKUTA, Matúš. *Zjištění identity prohlížeče pomocí WebAssembly.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Libor Polčák, Ph.D.

Zjištění identity prohlížeče pomocí WebAssembly

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Libora Polčáka. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

. . . . . .
Matúš Škuta
28. května 2020

Poděkování

Ďakujem Ing. Liborovi Polčákovi Ph.D. za vedenie, odbornú pomoc a rady pri vypracovaní bakalárskej práce.

Obsah

1 Úvod **3**

2 Zoznámenie sa s pojmami **4**
2.1 Identifikácia zariadení 4
2.1.1 Identifikácia zariadenia pomocou canvas elementu 4
2.1.2 Identifikácia pomocou audio kontextu 5
2.2 HTTP hlavičky 6
2.2.1 Odhad prehliadačov podľa dostupných HTTP hlavičiek 8
2.3 Detekcia dostupných rozšírení 8
2.4 Zabraňovanie identifikácie zariadenia 9
2.4.1 JS Restrictor 10
2.4.2 Rozšírenie Web API Manager pre prehliadač Firefox 10
2.4.3 Prehliadače 10
2.4.4 Zabraňovanie identifikácie pomocou canvas elementu 11

3 Web Assembly **12**
3.1 Pamäť Web Assembly 12
3.2 Emscripten a knižnica emscripten 12
3.3 Web Assembly a Javascript 14
3.4 Obmedzenia Web Assembly 14

4 Návrh **15**
4.1 Návrh programu 15
4.2 Návrh metód na identifikáciu zariadení 15
4.3 Obmedzenia 18

5 Implementácia **19**
5.1 Implementácia programu 19
5.1.1 Klient (frontend) 20
5.1.2 Web Assembly modul a implementované metódy identifikácie v module 21
5.2 Zábrany v implementácii identifikačných metód 24
5.3 Porovnanie webových stránok na získanie informácií o zariadení 24

6 Testovanie a obmedzenia implementácie **26**
6.1 Testovanie metódy identifikácie grafického hardwaru 26
6.2 Testovanie implementácie rozšírením Web API Manager v prehliadači Firefox 27
6.3 Testovanie metód identifikácie rýchlosti koliečka myši a rozlíšenia displeja . 28
6.4 Testovanie metódy identifikácie blokerov reklám 28

6.5 Testovanie implementácie rozšírením JS Restrictor 28
6.6 Testovanie prístupu Web Assembly funkcií k vstavaným objektom 29

7 Záver **30**

Literatura **31**

Kapitola 1

Úvod

V tejto práci sa budem zaoberať identifikáciou zariadení s možnosťou pripojenia na internet pomocou webových prehliadačov. Každý webový prehliadač poskytuje obrovské množstvo informácií, ktorých kombináciu budeme využívať na identifikovanie zariadenia.

Existuje obrovské množstvo nástrojov a techník na zabránenie identifikácie odtlačku zariadenia rôznymi spôsobmi, ktoré si spomenieme v časti 2.4. My sa budeme snažiť tieto jednotlivé nástroje a techniky obísť za pomoci novej technológie, ktorou je Web Assembly (priblížime si ju v kapitole 3). Emscripten, ktorý si predstavíme v časti 3.2, umožňuje kompiláciu binárnych jazykov do Web Assembly, čo vytvorí efektívny a rýchly program, ktorý sa svojou efektivitou a rýchlosťou rovná originálnemu strojovému kódu.

Cieľom tejto práce bude implementovať webovú stránku na identifikáciu zariadení pomocou technológie Web Assembly a otestovať implementáciu, oproti stávajúcim nástrojom a technikám na zabránenie identifikácie zariadení. Budeme sa hlavne zamerievať na prehliadač Brave, ktorý si priblížime v časti 2.4.3, rozšírenie JS Restrictor (viacej sa dozvieme v časti 2.4.1) a rozšírenie pre prehliadač Firefox Web API Manager (viď časť 2.4.2).

Identifikácia zariadení sa už môže klasifikovať ako zásah do súkromia osôb. V podstate sa jedná o identifikáciu zariadení, ktorá môže viesť k identifikácii užívateľa [7]. Preto Európsky výbor pre ochranu údajov, doporučuje implementáciu potvrdzovacieho formulára, kde užívateľ bude povolovať prístup k dátam, ktoré prehliadač poskytuje webovým stránkam - viacej v sekcii 2.1 [2].

V kapitole 2, si vysvetlíme čo je to identifikácia zariadení a predstavíme si niekoľko najpoužívanejších metód identifikácie zariadení. Tak isto si vysvetlíme jednotlivé HTTP hlavičky, ktoré budeme využívať pri implementácii a zoznámime sa s metódami a rozšíreniami na zabránenie identifikácie zariadení. V kapitole 3 sa zoznámime s technológiou Web Assembly a knižnicou Emscripten, pomocou ktorej budeme kompilovať C kód do Web Assembly. Tak isto si hlbšie rozoberieme ako Web Assembly a knižnica Emscripten funguje. V kapitole 4 si predstavíme návrh programu a jednotlivé návrhy metód, ktoré sa pokúsime implementovať. V kapitole 5 si rozoberieme implementáciu webovej stránky na identifikáciu zariadení, priblížime si viaceré metódy identifikácie, ktoré sme implementovali a vysvestlíme si, prečo niektoré metódy nie sme schopný implementovať. V kapitole 6, sa otestuje naša implementácia voči stávajúcim metódam, rozšíreniam a prehliadačom na zabránenie identifikácie zariadení.

Kapitola 2

Zoznámenie sa s pojmami

V tejto kapitole si vysvetlíme jednotlivé pojmy, s ktorými sa budeme stretávať v tejto práci. Vysvetlíme si pojmy ako je identifikácia zariadení, predstavíme si rôzne metódy identifikácie zariadení, rozšírime si pojmy o jednotlivých HTTP hlavičkách, s ktorými budeme pracovať a nakoniec si predstavíme existujúce spôsoby, programy a rozšírenia na zabraňovanie identifikácie zariadení.

2.1 Identifikácia zariadení

Identifikácia zariadení predstavuje spôsob sledovania a identifikácie užívateľov na internete pomocou získania množstva informácii o zariadení. Následná kombinácia týchto informácií sa využije na identifikáciu zariadenia, ktoré užívateľ používa. Každá získaná informácia obsahuje odlišný počet identifikačných informácii, ktoré môžu viesť k celkovej identifikácii zariadenia. Preto je potrebné tieto informácie vedieť rozoznať a vložiť ich do preddefinovanej tabuľky informácii, kde každá informácia má svoje miesto. Tabuľka sa následne pomocou dostupnej hašovacej funkcie (MD5, SHA-1) zahašuje a výsledný zahašovaný reťazec bude reprezentovať zariadenie, ktoré sme sa snažili identifikovať.

Identifikácia zariadení umožňuje identifikovať prehliadač a následne sledovať správanie užívateľov na internete - ktoré stránky navštevujú, ako dlho sa na daných stránkach zdržiavajú a mnoho ďalších dát, z ktorých je možné vytvoriť históriu navštívených stránok na internete, popularitu navštívených stránok a mnoho ďalších. To nás privádza na otázku súkromia na internete, ktoré je touto metódou narušované. Z tohto dôvodu bolo navrhnuté skupinou Euróspkeho výboru pre ochranu údajov, aby sa implementovala forma povolenia, ktorá užívateľovi pridá kontrolu nad tým či bude dovolovať webovým stránkam získavanie informácii. Jedným z konkrétnych navrhnutých riešení implementácie, bolo využítie Do Not Track hlavičky, ktorá by bola pridávaná k HTTP požiadavke a zakazovala by webovým stránkam získavanie informácii o zariadení [2].

2.1.1 Identifikácia zariadenia pomocou canvas elementu

WebGL API[1] je JavaScript API na renderovanie vysoko-výkonných 3D a 2D grafických scén za pomoci ktoréhokoľvek prehliadača bez použitia rozšírení, ktorý je kompatibilný s WebGL API. WebGL API je implementované tak, aby úzko zodpovedalo OpenGL ES 2.0[2]. WebGL API priamo pracuje s užívateľským grafickým hardwarom.

[1] WebGL API https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API
[2] OpenGL ES 2.0 https://www.khronos.org/registry/OpenGL-Refpages/es2.0/

Identifikácia grafického zariadenia pomocou canvas elementu pracuje s WebGL API, ktoré na základe požiadaviek na vykreslenie textu s efektami zvolenými na zvýšenie jedinečnosti, vyrenderuje obrázok s textom, ktorý je špecifický pre grafický hardware. Získaný obrázok sa premení na reťazec, ktorý bude slúžiť na identifikáciu grafického zariadenia. Reťazec sa využíje v kombinácii s ďalšími informáciami na celkovú identifikáciu zariadenia, alebo prehliadača.

Ako je možné vidieť v ukážke kódu 2.1 tak nám stačí iba 9 riadkov JavaScript kódu na identifikáciu grafického zariadenia pomocou canvas elementu. Ukážka kódu nám do canvas elementu vykreslí jeden oranžový štvorec do hornej časti canvas elementu. Spolu zo štvorcom sa vykreslí text nad štvorec, ktorému sa priradí mixovanie farieb tak, aby sa zvýšil rozdiel v renderovaní.

```
const txt = "BrowserLeaks,com <canvas> 1.0";
const canvas = document.createElement('canvas');
const context = canvas.getContext('2d');
context.textBaseline = "top";
context.font = "14px 'Arial'";
context.textBaseline = "alphabetic";
context.fillStyle = "#f60";
context.fillRect(125,1,62,20);
context.fillStyle = "#069";
context.fillText(txt, 2, 15);
context.fillStyle = "rgba(102, 204, 0, 0.7)";
context.fillText(txt, 4, 17);
const res = canvas.toDataURL();
```

Výpis 2.1: Ukážka JavaScript kódu na identifikáciu pomocou canvas elementu prevzaná z návodu [1]

2.1.2 Identifikácia pomocou audio kontextu

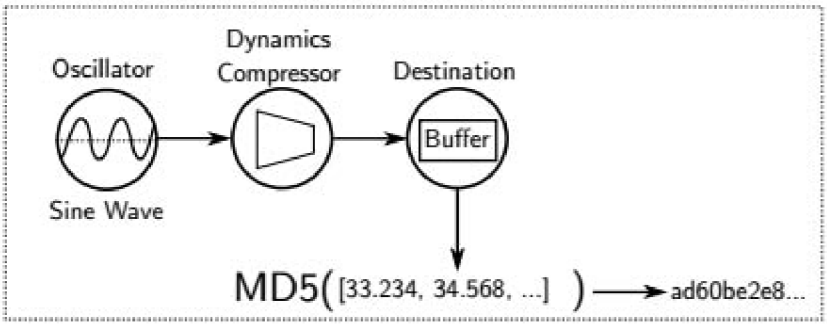
Identifikácia pomocou audio kontextu je technika, na identifikáciu zariadení, alebo prehliadačov pomocou identifikácie jemných odlišností v renderovaní statickej zvukovej vlny pomocou audio API. Metóda identifikácie pomocou audio kontextu pozostáva z dvoch metód identifikácie zariadení alebo prehliadačov [3].

Prvá metóda [3] využíva na identifikáciu audio kontext v režime offline, ktorý vygenerovaný zvuk neprehráva, ale iba ukladá do vyrovnávacej pamäte, ako je možné vidieť na obrázku 2.1. Metóda využíva oscilátor node[3], ktorý vygeneruje sínusovú, alebo trojuholníkovú vlnu, ktorú následne predá nodu dynamického kompresora[4]. Node dynamického kompresora upraví zvuk na základe prednastavených hodnôt, ktoré sa líšia podľa zvukovej karty a prehliadača. Upravený zvuk je uložený do vyrovnávacej pamäte, ktorá sa následne predá hašovacej funkcii, ktorá bude reprezentovať odtlačok audio kontextu.

Druhá metóda [3] využíva na identifikáciu audio kontext v online režime, ktorý priamo prehráva vygenerovaný zvuk. Znova ako pri prvej metóde využijeme oscilátor node na vygenerovanie sínusovej alebo trojuholníkovej vlny. Vygenerovaná vlna sa následne predá

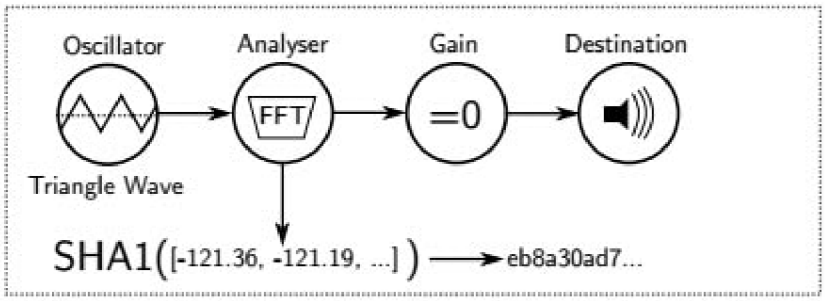
[3]Oscilátor Node https://developer.mozilla.org/en-US/docs/Web/API/OscillatorNode

[4]Node dynamického kompresora https://developer.mozilla.org/en-US/docs/Web/API/DynamicsCompressorNode



Obrázek 2.1: Ukážka identifikácie pomocou audio API v režime offline prevzatá z návodu [3]

analyzér nodu[5]. Analyzér node prevedie na vlne rýchlu furierovu transformáciu z prednastavenými hodnotami. Prednastavené hodnoty sa líšia podľa prehliadača, alebo zvukovej karty. Výsledok transformácie signálu z node analyzéru sa napojí na gain node, ktorý stlmí zvuk, aby užívateľ nemusel počúvať nepríjemný zvuk, ktorý sme vygenerovali. Výsledok transformácie, ktorý sme predali gain nodu tak isto predáme skriptProcesor nodu[6]. SkriptProcesor node obsahuje možnosť zahlásenia sa na udalosť onaudioprocess, ktorú využijeme na získanie výsledku rýchlej furierovej transformácie. Po každom spustení eventu získame časť výsledku, ktorú začneme postupne vkladať do vyrovnávacej pamäte. Keď sa vo vyrovnávacej pamäti nachádza dopredu preddefinovaná dĺžka dát, predáme vyrovnávaciu pamäť hašovacej funkcii, ktorá bude reprezentovať odtlačok audio kontextu.



Obrázek 2.2: Ukážka identifikácie pomocou audio API v režime online prevzatá z návodu [3]

2.2 HTTP hlavičky

HTTP hlavičky definujú formu dát a využívajú sa na komunikáciu medzi serverom a klientom. Hlavičky HTTP požiadavky obsahujú informácie o požadovaných dátach a požadovanej odpovedi, ktorej vie klient porozumieť. Informácie hlavičky nedoplňuje priamo užívateľ, ale sú doplňované automaticky prehliadačom. My sa budeme zaoberať hlavičkami, ktoré obsahujú informácie o klientovi, zariadení klienta z ktorého sa požiadavka zaslala a mnoho ďalších. Z týmito rôznymi hlavičkami sa zoznámime v nasledujúcich vysvetleniach.

[5]Analyzér Node https://developer.mozilla.org/en-US/docs/Web/API/AnalyserNode
[6]SkriptProcesor Node https://developer.mozilla.org/en-US/docs/Web/API/ScriptProcessorNode

Hlavička Accept[7] obsahuje typ obsahu (Content-Type[8]), ktorý prehliadač požaduje a je mu schopný porozumieť. Môže sa jednať o viaceré typy, z ktorých si server jeden vyberie a odpovie vybraným typom odpovede. Hlavička môže vyzerať napríklad takto: `Accept: text/html, application/xhtml+xml, application/xml;q=0.9, image/webp, */*;q=0.8`.

Hlavička Accept-Encoding[9] zašle viaceré algoritmy na kompresiu dát, ktoré prehliadač podporuje. Server vyberie jeden z poskytnutých algoritmov a v odpovedi vyberie najlepší možný algoritmus. Hlavička môže vyzerať napríklad takto: `Accept-Encoding: deflate, gzip;q=1.0, *;q=0.5`.

Hlavička Accept-Language[10] reprezentuje jazyk, ktorému je užívateľ schopný porozumieť. Obvykle sa jedná o jazyk, ktorý je nastavený pre daný prehliadač. Hlavička obvykle vyzerá napríklad takto: `Accept-Language: fr-CH, fr;q=0.9, en;q=0.8, de;q=0.7, *;q=0.5`

Hlavička User-Agent[11] je reťazec, ktorý pozostáva z identifikácie aplikácie, operačného systému a ostatných identifikačných dát. Hlavička User-Agent môže vyzerať napríklad takto: `Mozilla/5.0 (User-AgentWindows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.88 Safari/537.36`

Hlavička Referer[12] je reťazec, ktorý môže obsahovať adresu webovej stránky na ktorej sa užívateľ nachádzal pred tým, než navštívil našu webovú stránku, tak isto môže reprezentovať mobilnú aplikáciu pomocou ktorej sa načítala naša webová stránka. Hlavička Referer môže vyzerať napríklad takto: `Referer: https://l.facebook.com/`, čo reprezentuje presmerovanie zo stránky Facebook na našu webovú stránku.

Hlavička dnt (Do Not Track)[13] je obvykle hodnota nadobúdajúcu pravdivostnú hodnotu 0 alebo 1 ale môže byť rozšírená o ďalšie informácie. Jednotka v tomto kontexte reprezentuje požiadavku prehliadača, aby užívateľ nebol sledovaný. Jedná sa iba o požiadavku klienta na webovú stránku a preto dosť reklamných spoločností túto požiadavku ignoruje a naďalej sleduje užívateľa. Ignorujú ju hlavne preto, lebo nie je po nich požadované, aby hlavičku DNT dodržovali.

Hlavičky Sec-Fetch[14] reprezentujú zabezpečené typy hlavičiek, čo znamená že nemôžu byť menené pomocou Javascriptu. Hlavičky reprezentujú metadáta o kontexte, ktorý stránka požaduje od serveru. Hlavičky Sec-Fetch pozostávajú zo 4 typov hlavičiek:

[7] Hlavička Accept https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept
[8] Content-Type https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type
[9] Hlavička Accept-Encoding https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Encoding
[10] Hlavička Accept-Language https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Language
[11] Hlavička User-Agent https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent
[12] Hlavička Referer https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer
[13] Hlavička DNT https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/DNT
[14] Sec-Fetch hlavička https://www.w3.org/TR/fetch-metadata/

Sec-Fetch-Dest[15] je hlavička reprezentujúca typ dát, ktoré prehliadač požaduje. Môže sa jednať o dokument, zvuk, obrázok alebo rôzne ďalšie typy, ktoré dokáže prehliadač spracovať. Hlavička Sec-Fetch-Dest môže vyzerať napríklad takto: `Sec-Fetch-Dest: document`.

Sec-Fetch-Mode[16] je hlavička reprezentujúca typ požiadavky, ktorý vyžadujeme od serveru. Hlavička Sec-Fetch-Mode môže vyzerať napríklad takto: `Sec-Fetch-Mode: same-origin`.

Sec-Fetch-Site[17] je hlavička reprezentujúca vzťah medzi pôvodom klienta, ktorý vytvára požiadavku a pôvodom serveru, ktorý vytvorenú požiadavku prijíma. Hlavička Sec-Fetch-Site môže vyzerať napríklad takto: `Sec-Fetch-Site: same-site`.

Sec-Fetch-User[18] je hlavička, ktorej hodnota je reprezentovaná ako pravdivostná hodnota. Pravdivostná hodnota reprezentujúca, či bola požiadavka vygenerovaná vstupom od užívateľa, alebo bola vygenerovaná programom. Hlavička Sec-Fetch-User môže vyzerať napríklad takto: `Sec-Fetch-User: ?0`.

2.2.1 Odhad prehliadačov podľa dostupných HTTP hlavičiek

Odhad prehliadačov bude prebiehať pomocou už spomenutých HTTP hlavičiek. Po dôkladnej analýze HTTP hlavičiek vygenerovaných prehliadačmi, na ktorých budeme testovať aplikáciu a ktoré si spomenieme v úvode kapitoly 6 sme zistili, že môžeme kategorizovať prehliadače a zariadenia, na ktorých dané prehliadače pracujú do dvoch skupín.

Prvá skupina nezasiela HTTP hlavičky typu Sec-Fetch. Skupina pozostáva z prehliadačov Firefox, Safari a Apple zariadení. Všetky prehliadače, ktoré pracujú na zariadení Apple nepoužívajú Sec-Fetch hlavičky.

Druhá skupina využíva HTTP hlavičky typu Sec-Fetch. Jedná sa o prehliadače Google Chrome, Microsoft Edge, Opera a prehliadač Brave. Tieto prehliadače vychádzajú z projektu Chromium[19].

2.3 Detekcia dostupných rozšírení

Rozšírenia[20] sú malé programy, ktoré upravujú funkcionalitu prehliadačov a správania prehliadačov pre individuálne potreby a preferencie. Sú postavené na webových technológiach ako je HTML, CSS a JavaScript. Sú obvykle inštalované užívateľmi a sú zneužiteľné k identifikácii užívateľov, pretože užívatelia používajú špecifickú kombináciu rozšírení.

Detekcia dostupných rozšírení je možná hlavne z toho dôvodu, že veľa rozšírení pridáva funkcionalitu, správanie, alebo priamo upravuje webovú stránku. Detekcia týchto pridaných možností je obtiažna, pretože z Web Assembly nemáme priamy prístup k objektom, ktoré odkazujú na rozšírenú funkcionalitu. Preto si v ďalších podsekciách priamo vysvetlíme, akým spôsobom budeme zisťovať dostupné rozšírenia [13].

Detekcia reklamných blokerov

Reklamné blokery ako AdBlock Plus[21] a uBlock pracujú na dvoch úrovniach - sieťovej a dokumentovej.

[19] Projekt Chromium https://www.chromium.org/
[20] Rozširenia https://developer.chrome.com/extensions
[21] AdBlock Plus https://adblockplus.org/

Dokumentová úroveň prehľadáva DOM (Document Object Model) strom, kde sa snaží nájsť elementy s triedami, alebo identifikátormi, ktorých názov, alebo časť názvu sú všetky možné kombinácie slova advertisement (reklama). Po nájdení takéhoto elementu sa daný element spolu s jeho detskými elementami odstráni z DOM stromu, čím sa zablokuje zobrazovanie alebo načítavanie reklamy.

Sieťová úroveň kontroluje zasielané HTTP požiadavky na domény. Domény porovnáva so svojimi zoznamami reklamných domén a po nájdení zhody sa požiadavka zablokuje, aby sa nemohol kontaktovať reklamný server.

Reklamné blokery využíva v roku 2020 približne 26.4% užívateľov v Amerike [9] čo je množina užívateľov, ktorá môže slúžiť ako ďalší identifikačný parameter celkovej identifikácie zariadenia.

2.4 Zabraňovanie identifikácie zariadenia

Na zabránenie identifikácie zariadení na internete bolo vytvorených viacero stratégii:

Blokovanie skriptov je pomerne efektívne, no relatívne často zabraňuje bežnej funkcionalite stránky. Predstaviteľom tejto stratégie je rozšírenie NoScript[22] pre prehliadače Firefox a Chrome, ktorý blokuje vykonávanie skriptov na webových stránkach, čím blokuje množstvo metód identifikácie zariadení [17].

Blokovanie parametrov, ktoré ako názov udáva blokuje parametre poskytujúce informácie na identifikáciu zariadení a pri ich vyžiadaní vráti `undefined`. Táto stratégia je veľmi ľahko rozpoznateľná, ale je ju zasa možné aj veľmi ľahko implementovať [11].

Pridávanie šumu k atribútom pridáva náhodné hodnoty k parametrom, alebo zaokrúhli jednotlivé hodnoty tak, aby nebolo môžné ich s presnosťou určiť. Táto technológia je už implementovaná vo viacerých prehliadačoch, viac si o nej povieme v časti 2.4.3.

Rekonfigurácia pomocou virtualizácie, kde sa pomocou virtualizácie vytvorí odlišné zariadenie od nášho, ktorého informácie sa budú zobrazovať namiesto reálneho zariadenia. Túto stratégiu je veľmi ťažké odhaliť pokiaľ je dobre implementovaná. Ak sa ale jedná o niektoré virtuálne stroje upravujú reťazec, ktorý reprezentuje v hlavičke User-Agent (ktorú sme si spomenuli v časti 2.2, odstavec o hlavičke User-Agent) operačný systém a pridávajú mu svoju vlastnú hodnotu, čo môže viesť k odhaleniu využitia virtualizácie.

Zmenenie atribútov za existujúce hodnoty zmení hodnoty tak, aby sa zariadenie z ktorého pristupujeme na internet vydávalo za odlišné zariadenie. Táto stratégia je veľmi podobná so stratégiou virtualizácie bez využitia virtualizácie. Hlavným problémom tejto stratégie je implementácia v reálnom svete, keďže je skoro nemožné ju správne implementovať [11]. Je to hlavne z toho dôvodu že existuje množstvo parametrov, z ktorých je možné identifikovať zariadenia. Webové štandardy sa neustále upravujú pridávajú sa nové API, ktoré môžu obsahovať nové parametre ktoré sa môžu využiť na identifikáciu prehliadača.

Ďalej si predstavíme niekoľko programov, ktoré tieto metódy využívajú [16].

[22] NoScript Extension https://noscript.net/

2.4.1 JS Restrictor

JS Restrictor je rozšírenie, ktoré ako názov udáva pracuje na úrovni Javascriptu a upraví funkcionalitu dostupných Javascriptových API a funkcii, aby čo najviac obmedzil množstvo získaných dát o zariadení pri pokuse o identifikáciu zariadenia a zároveň sa snaží, aby neobmedzoval užívateľa pri prehliadaní webových stránok.

JS Restrictor prepisuje dostupné API a na ich úpravu využíva tri stratégie na zmeny výsledkov volaní. Prvou stratégiou je vracanie falošných dát, slúžiacich na oklamanie webovej stránky, ktorá sa snaží o identifikáciu zariadenia. Druhou stratégiou je pridávanie šumu k atribútom, ktoré vracajú presné dáta. Treťou stratégiou je blokovanie parametrov, kde sa pri vyžadovaní parametrov vráti `null` alebo `undefined`.

JS Restrictor pracuje so štyrmi vstavanými úrovňami bezpečnosti, na základe ktorých si môže užívateľ nastaviť koľko informácii môže mať webová stránka dostupných o prehliadači, alebo tiež o zariadení, na ktorom práve prehliadač beží.

JS Restrictor umožňuje nastaviť vlastný level bezpečnosti, kde si užívateľ môže zvoliť čo všetko chce užívateľ zmeniť. JS Restrictor poskytuje možnosť nastavenia hlavičky User Agent (viď časť 2.2, odstavec o hlavičke User-Agent), aby sa vydávala za za prehliadač Google Chrome, taktiež umožňuje zmenu hlavičky Accept-Language (viď časť 2.2, odstavec o hlavičke Accept-Language) na hodnotu `en-US,en;q=0.5`, aby sa vydávala za amerického návštevníka stránky. Ďalej umožňuje vymazávanie hlavičky Referer (viď časť 2.2, odstavec o hlavičke Referer), aby sa navštívená stránka nemohla dozvedieť odkiaľ sme na webovú stránku prišli. Takisto sa snaží o zabránenie identifikácie pomocou Canvas elementu tým spôsobom, že upraví funkciu, pomocou, ktorej sa vracajú vykreslené dáta, aby vracala statické dáta. Tak isto umožňuje zredukovanie presnosti času a geolokačných dát.

2.4.2 Rozšírenie Web API Manager pre prehliadač Firefox

Rozšírenie Web API Manager[23] je vytvorené iba pre prehliadač Firefox a umožňuje zakazovanie webových API na zabezpečenie a urýchlenie prehliadania webových stránok. Web API Manager umožňuje vypínať podporu webových API v prehliadači Firefox. Je možné deaktivovať väčšinu dostupných rozhraní vrátane rozhrania na využitie viacerých jadier počítača (WebWorker), API pre základnú funkcionalitu webov, grafické API, API na interakciu z hardwarom a senzormi zariadenia, API na sieťové operácie. Umožňuje nastaviť pravidlá blokovania viacerých API pre konkrétne webové stránky. Zakázané API ovplyvní ako Javascript, tak isto aj novú technológiu Web Assembly (viď kapitola 3).

2.4.3 Prehliadače

Prehliadače Firefox[24] a Chrome[25] majú implementované zaokrúhľovanie výsledku funkcie `performance.now()` [8], aby zabránili časovým útokom a pokusom o identifikáciu zariadení na základe času.

Prehliadač Chrome má v pláne do budúcna implementovať spôsob obmedzenia formou rozpočtu. To znamená, že bude vracať len toľko informácii, aby identifikácia zariadenia nedokázala identifikovať jednotlivé zariadenia, ale aby dokázala iba zaradiť dané zariadenie alebo prehliadač do rozradzovacích skupín použiteľných na reklamné a marketingové účely [12].

[23] Rozšírenie Web API Manager https://addons.mozilla.org/en-US/firefox/addon/webapi-manager/
[24] Prehliadač Firefox https://www.mozilla.org/sk/firefox/new/
[25] Prehliadač Chrome https://www.google.com/intl/sk/chrome/

Prehliadač Tor Browser[26] využíva hlavne stratégiu, pri ktorej zmení atribúty za už existujúce hodnoty a túto stratégiu využíva pre všetkých svojich užívateľov tak, aby každý z nich vyzeral z pohľadu webovej stránky snažiacej sa o identifikáciu zariadenia úplne totožne. Ďalej pri pokuse o identifikáciu pomocou canvasu využíva statickú metódu, kde pri pokuse o čítanie z canvasu vracia statickú hodnotu. Viacej si spomenieme o tejto stratégii v časti 2.4.4. Z pohľadu blokácie identifikácii zariadení je prehliadač Tor Browser pravdepodobne najlepšou možnosťou, kde sa riziko identifikácie zariadenia znižuje s pribúdajúcim počtom užívateľov.

Prehliadač Brave[27] blokuje viaceré metódy identifikácii zariadení. Zabraňuje identifikácii pomocou canvas elementu, ako je vysvetlené v časti 2.4.4. Brave pracuje na veľmi podobný spôsob ako prehliadač Tor - podľa mojich vlastných zistení sa snaží vydávať za odlišný prehliadač tak, ako bolo spomenuté i v jeho dokumentácii [4]. Popri tom ako prehliadač Brave blokuje identifikáciu zariadení, má vo svojom jadre implementovanú aj blokáciu reklám.

Mobilná verzia prehliadača Brave sa snaží pracovať na podobnom princípe ako jeho počítačová verzia ale bohužiaľ je veľmi ovplyvňovaná aktualizáciami WebView v systéme Android, ktorý pridáva nové, alebo upravuje už existujúce webové štandardy. Tak isto je mobilný prehliadač Brave limitovaný mobilnými systémami, ktoré limitujú tvorcov prehliadača viac než sú limitovaní na počítačoch. Napriek týmto limitáciam sú tvorcovia schopní implementovať verziu mobilného prehliadača Brave, ktorá sa na limitovaný čas podobá počítačovej implementácii. Z týchto dôvodov je veľmi obtiažna identifikácia mobilného prehliadača Brave.

2.4.4 Zabraňovanie identifikácie pomocou canvas elementu

Zabraňovanie identifikácie pomocou canvas elementu je možné pomocou troch typov stratégii.

Prvá stratégia pozostáva z vypnutia canvas funkcionality, čo znamená, že jediná identifikácia, ktorú môže webová stránka zistiť je nedostupnosť canvas funkcionality.

Druhá stratégia upraví canvas funkcionalitu tak, aby pri každej požiadavke na vykreslenie sa reálne vyrenderovalo všetko čo je požadované. Úprava nastáva až pri konvertovaní canvas elementu na bajtový reťazec. Tu je funkcia na konverziu obrázka na reťazec upravená tak, aby vracala odlišné RGB hodnoty. Vrátené hodnoty môžu byť buď statické, kde sa vráti iba jedna staticky nastavená farba (obvykle to býva biela farba), alebo sa vráti statický predrenderovaný obrázok pre všetkých užívateľov programu, ktorý túto stratégiu implementuje.

Tretia stratégia tak isto upravuje canvas funkcionalitu tým spôsobom, že k výsledku pridá pseudo-náhodné hodnoty, ktoré upraví nepatrným spôsobom tak, aby užívateľovo oko nepoznalo rozdiel. Táto stratégia síce nenarušuje vykresľovanie na webových stránkach, ale nie je ani veľmi efektívna proti identifikácii pomocou canvasu, pretože iba pridáva určité hodnoty k výsledku, ktoré stále ostávajú jedinečné [11].

[26] Prehliadač Tor Browser https://www.torproject.org/download/
[27] Prehliadač Brave https://brave.com/

Kapitola 3

Web Assembly

Web Assembly[1] je binárny inštrukčný formát pre zásobníkový typ virtuálneho stroja. Bol navrhnutý tak aby bol prenosný, rýchly, pamäťovo bezpečný a vhodný na kompilovanie jazykov, ako sú C/C++/Rust do binárnej formy ktorá sa skonvertuje na Web Assembly. To dovoľuje implementáciu klientskych a serverových aplikácii na webe aj pre ľudí, ktorí neovládajú JavaScript. Rýchlosť spracovania Web Assembly je porovnateľná s rýchlosťou spracovania vygenerovaného binárneho formátu z jazykov C/C++ a Rust. Kód je možné priamo písať vo Web Assembly, no nie je to doporučované, pretože je to podobné písaniu kódu v Assembly jazyku [6].

3.1 Pamäť Web Assembly

Pamäť Web Assembly je reprezentovaná ako JavaScript objekt obsahujúci pole bytov, ku ktorému má prístup Web Assembly a takisto aj JavaScript. O pamäť sa stará garbage collector, ktorý po ukončení modulu automaticky uvolní pamäť, aby sa vyhol únikom pamäte [5]. Pamäťové pole je môžné zväčšovať počas behu programu. Ďalej je umožnené si predávať hodnoty pomocou pamäťového pola medzi JavaScriptom a Web Assembly prostredníctvom indexu v poli pamäťe, ktoré sa referencuje medzi Web Assembly a JavaScriptom. Zdielanú pamäť využívame pri zobrazovaní zistených informácii o zariadení do HTML tabuľky.

Pri indexovaní pamäťe je kladený veľký dôraz na kontrolu indexu, teda či sa daný index nachádza vo vymedzenej pamäti programu. Ak sa pokúsime indexovať mimo pole, tak Web Assembly vyvolá výnimku.

3.2 Emscripten a knižnica emscripten

LLVM[2] je infraštruktúra pre prekladač navrhnutá primárne pre C/C++ jazyky, dokáže ale pracovať aj s inými jazykmi. Využíva sa na kompiláciu jazykov do binárnej formy.

Emscripten je kompiler LLVM na JavaScript, alebo Web Assembly, ktorý je Open Source. Dokáže skonvertovať každý jazyk, ktorý môže byť skompilovaný do binárneho kódu pomocou LLVM.

Emscripten knižnica pracuje s WebGL pomocou OpenGL ES 2.0, ktorého príkazy sú namapované priamo na WebGL. Umožňuje pracovať s obidvoma WebGL verziami. Verzia

[1] Web Assembly https://webassembly.org/
[2] Low Level Virtual Machine http://llvm.org/

WebGL sa volí pri kompilovaní do Web Assembly. Emscripten knižnica takisto obsahuje viaceré API, ktoré budeme využívať pri identifikácii zariadení.

HTML5 API[3] definuje nízkoúrovňovú interakciu s HTML5 eventami, priamo z C a C++ kódu. Neumožňuje priamu interakciu s HTML DOM elementom, ale umožňuje interakciu s window, document, screen a canvas elementami, ktorým môže pridávať eventy a interagovať s nimi s pomocou určitých funkcii. HTML5 API obsahuje interakciu z API na kontrolu vibrácii a API na zistenie stavu batérie, ktoré bohužiaľ v danej dobe nefungujú.

API obsahuje viaceré mini API, ktoré budeme ďalej využívat pri implementácii. Niektoré z nich sú:

API na orientáciu zariadenia[4] umožňuje získať náklon zariadenia v 3D priestore. Náklon je reprezentovaný tromi súradnicami X, Y a Z, reprezentujúce rotáciu na daných osách. API na orientáciu zariadenia pracuje takým spôsobom, že keď zariadenie neobsahuje gyroskop, alebo má prístup k nemu zablokovaný tak stále vracia súradnice X,Y a Z, ktorých hodnoty sú nastavené na 0 a preto nie sme schopní identifikovať či sa jedná o zariadenie z dostupným gyroskopom.

API na pohyb zariadenia[5] umožňuje získať smer pohybu zariadenie v priestore 3D. Pohyb je reprezentovaný tromi súradnicami X, Y a Z, reprezentujú smer pohybu zariadenia v metroch. Súradnice X,Y a Z je možné získať už zo započítanou gravitáciou, alebo bez gravitácie. API na pohyb zariadenia pracuje na veľmi podobný princíp ako API na orientáciu zariadenia v tom zmysle že obe API pri vypnutej funkcionalite, alebo pri nedostupnosti gyroskopu nastavia hodnoty súradnic X, Y, Z na hodnotu 0.

Preable.js[6] dovoľuje prístup ku skompilovanému C kódu, ku ktorému je priložené volanie funkcii jazyka C, prístup k pamäti a konverzia C reťazových ukazovateľov na JavaScript reťazec (String) a tiež umožňuje pracovať z reťazovým ukazovateľom s rôznymi formátmi a kódovaniami.

Audio[7] v Emscripten knihovni obsahuje implementáciu OpenAL 1.1 API, kde využíva Web Audio API ako backend. Bohužiaľ emscripten audio neobsahuje API na interakciu z OpenAL 1.1 alebo Web Audio API ale sľubujú že do budúcna sa ju budú snažiť implementovať. Aj keď audio API nie je dostupné je tu stále možnosť pracovať z C audio knihovňami, ktoré sa namapujú na OpenAL. Problémom je však implementácia identifikácie pomocou audio kontextu (spomenuli sme si ju v predchádzajúcej časti 2.1.2), ktorý si vysvetlíme v časti 5.2.

[3] HTML5 API implementované v `html5.h` https://github.com/emscripten-core/emscripten/blob/master/system/include/emscripten/html5.h

[4] Device orientation API https://emscripten.org/docs/api_reference/html5.h.html#device-orientation

[5] Device motion API https://emscripten.org/docs/api_reference/html5.h.html#device-motion

[6] Preamble.js https://emscripten.org/docs/api_reference/preamble.js.html

[7] Emscripten Audio https://emscripten.org/docs/porting/Audio.html?highlight=audio

WebVR je technológia virtuálnej reality dostupná v prehliadačoch. Jedná sa už o zastaralú technológiu, ktorá bola hlavne dostupná v prehliadačoch Chrome a Firefox. WebVR API[8] poskytovalo základné rozhranie pre interakciu s WebVR priamo z kódu.

WebXR[9] je nové API nahradzujúce zastaralé API WebVR. WebXR API kombinuje virtuálnu realitu a rozšírenú realitu. WebXR je veľmi rozsiahle, pracuje z WebGL na renderáciu 3D objektov a scén. WebXR API je ešta stále vo vývoji a preto je iba z časti podporované prehliadačmi Chrome a Edge[10].

IndexDB API[11] jedná sa o jednoduché API na interakciu z IndexDB databázou. IndexDB je databáza uložená v prehliadači na klientskej strane. IdexDB pracuje z kľúčami ku ktorým priradí dátovú hodnotu. API sa skladá zo štyroch operácii z databázou. Kontrola existencie dát na základe kľúča, uloženie, získanie a vymazanie dát pomocou kľúča. Všetky tieto operácie sú asynchronné.

Fetch API[12] umožňuje prenos súborov pomocou XMLHttpRequest API. Podporuje požiadavky typu GET, PUT, POST, ale nepodporuje požiadavku typu DELETE. Prenášané súbory sú ukladané do IndexDB úložiska aby mohli byť prístupné pri pravidelných návštevách webovej stránky rýchlejšie. Fetch API je možné volať z viacerých vlákien aplikácie, a môžu byť volané synchrónne, alebo asynchrónne.

3.3 Web Assembly a Javascript

Web Assembly bolo navrhnuté s myšlienkou spolupráce s jazykom Javascript. Je tým hlavne myslené to, že Web Assembly nie je zatiaľ možné inicializovať bez pomoci Javascriptu. Tak isto rôzne Web Assembly funkcie využívajú Javascriptové funkcie. Niektoré spôsoby zisťovania času čo sú dostupné vo Web Assembly sa pri kompilácii prepíšu, aby získavali čas z Date objektu, ktorý ponúka Javascript a tým znížia presnosť času.

3.4 Obmedzenia Web Assembly

Ako sme si spomenuli v časti 2.4.3, kde sme sa dozvedeli, ako prehliadače implementujú rôzne techniky a obmedzenia na zabránenie identifikácií zariadení tak isto aj Web Assembly je obmedzované svojou implementáciou. Jedným z hlavných obmedzení je to, že Web Assembly pracuje vo virtualizovanom prostredí. Virtualizované prostredie nie je úplne oddelené od prehliadača, ale má definované svoje vlastné zvukové a grafické rozhrania, ktoré zapúzdrujú reálne rozhrania a tak sa nemôžme dostať priamo k nim. Tak isto pri inicializácii rôznych API, ktoré pri Javascripte majú nastavené predvolené hodnoty podľa prehliadačov alebo zariadenia, Web Assembly nastaví statické hodnoty pre všetky API nezávisle od prehliadača alebo zariadenia, na ktorom aplikácia beží.

[8] Emscripten WebVR library https://emscripten.org/docs/api_reference/vr.h.html#c.emscripten_vr_ready
[9] WebXR API https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API
[10] WebXR support https://caniuse.com/#search=webxr
[11] IndexDB Emscripten APIhttps://emscripten.org/docs/api_reference/emscripten.h.html#asynchronous-indexeddb-api
[12] Fetch API https://emscripten.org/docs/api_reference/fetch.html

Kapitola 4

Návrh

V tejto kapitole si vysvetlíme návrh programu na identifikáciu zariadenia, niekoľko metód na identifikáciu zariadení a vysvetlíme si jednotlivé obmedzenia, z ktorými sa budeme stretávať pri implementácii.

4.1 Návrh programu

Hlavná časť aplikácie bude slúžiť na extrakciu dát od užívateľa. Tieto získané dáta následne skombinuje tak, aby identifikovala zariadenie, z ktorého sa užívateľ pripojuje na webovú stránku.

Vedľajšia časť bude implementovaná ako webová stránka, ktorá bude načítať hlavnú časť aplikácie a spúšťať testovanie na pokyn užívateľa. Vedľajšia časť bude rozdelená na klient a server. Na serveri sa budú ukladať získané dáta z testov a bude využitý na prečítanie HTTP hlavičky. Na klientovi sa budú tieto testy spúšťať a následne zobrazovať ich výsledky, ako pre aktuálne testovacie zariadenie, tak aj pre už otestované zariadenia.

4.2 Návrh metód na identifikáciu zariadení

V nasledujúcich častiach si vysvetlíme návrhy jednotlivých metód, ktoré budú slúžiť na identifikáciu zariadenia.

Metóda identifikácie grafického hardwaru

Identifikácie grafického hardwaru bude prebiehať pomocou vykreslenia v canvas elemente pomocou WebGL. Na vykreslenie využijeme OpenGL API ktoré sa pri kompilácii premapuje do WebGL. Pre presnejšie výsledky by sa využilo vykreslenie textu v kombinácií z primitívnymi tvarmi. Bohužiaľ OpenGL API podporuje iba vykreslenie textu z dodaným fontom čo by mohlo skresliť výsledky identifikácie grafického hardwaru a preto vykreslíme iba jeden z primitívnych tvarov ktorým bude trojuholník. Pre zvýšenie jedinečnosti sa pridá prelínanie farieb na základe pozície, ktoré bude implementované vo fragment shadery. Viacej sa dozvieme v časti 5.1.2.

Metóda na identifikáciu počítača a mobilného zariadenia

Identifikácia počítača a mobilného zariadenia bude pracovať s viacerými možnosťami identifikácie, ktoré sa skombinujú a vyhodnotia, o aké zariadenie sa jedná.

Prvý typ identifikácie zariadenia bude pracovať s predpokladom, že mobilné zariadenia nepodporujú virtuálnu realitu. To môžeme otestovať pokusom o inicializáciu WebVR API (viz. časť 3.2), ktorého inicializácia pri mobilných zariadeniach zlyhá, keďže virtuálna realita nie je podporovaná. Pri počítačoch sa ale podarí, pretože by mala byť podporovaná.

Druhý typ identifikácie zariadenia bude slúžiť na zistenie orientácie zariadenia, na čo využíva dostupný gyroskop a akcelerometer na zistenie 3D orientačných súradníc. Gyroskop a akcelometer bývajú dostupné v mobilných zariadeniach a tabletoch, ale nie sú dostupné v počítačoch a notebookoch. Na zistenie dostupnosti akcelerometru a gyroskopu využíjeme HTML5 API (viz. časť 3.2), ktoré obsahuje eventy na zisťovanie orientácie zariadení[1]. Funkcia na zistenie orientácie zariadenia uspeje iba pri mobilných a tabletových zariadeniach.

Tretí typ identifikácie zariadenia bude slúžiť na zistenie dostupnosti batérie v zariadení[2], kde znovu využijeme HTML5 API obsahujúce eventy a funkcie na zisťovanie informácii o percentuálnom stave batérie, alebo ako dlho batéria vydrží. Tieto informácie slúžia na rozlíšenie stolných počítačov od mobilných zariadení, tabletov a notebookov.

Štvrtý typ identifikácie zariadenia slúži na zistenie dostupnosti vibrácií. To nám umožní lepšie rozoznať mobilné zariadenia od počítačov, pretože iba mobilné zariadenia majú v sebe zabudované vibračné motorčeky. Taktiež na to využijeme HTML5 API, ktoré obsahuje eventy a funkcie vibrácií[3], ktoré bohužiaľ nefunguje ako sa dozvieme v časti 5.2.

Nakoniec skombinujeme všetky získané informácie na približné odhalenie typu zariadenia. Môže sa jednať o mobilné zariadenia, notebooky, alebo stolné počítače, ktoré budeme schopní rozlíšiť. Viac je možné vidieť v tabuľke 4.1.

Tieto štyri testovania rozlišujú viackrát medzi tými istými zariadeniami z toho dôvodu, aby sme boli schopní odhaliť niektoré typy blokovania, ktoré sme si spomenuli v časti 2.4.

	WebVR	Orientácia zariadenia	Batéria	Vibrácie
Mobilné zariadenie/Tablet		✓	✓	✓
Stolný počítač	✓			
Notebook	✓		✓	

Tabulka 4.1: Tabuľka reprezentujúca, ako sa budú rozlišovať jednotlivé zariadenia

Metóda na získanie HTTP hlavičiek

Získanie informácií z GET požiadavky bude rozdelené na klientský a serverový kód.

Klientská časť sa postará o zaslanie HTTP GET požiadavky na server, ktorý bude serverovať klientskú aplikáciu. Zaslaná HTTP GET požiadavka sa automaticky vyplní prehliadačom dátami umožňujúcimi nám identifikovať prehliadač a zariadenie na ktorom prehliadač beží.

Serverová časť prijme a prečíta GET požiadavku, z ktorej nás budú zaujímať, najmä hlavičky pomocou ktorých budeme identifikovať zariadenie, ako bolo spomenuté v časti 2.2. Z požiadavky sa získa hlavička `Accept`, ktorá nám povie aké typy odpovedí prehliadač vie spracovať, hlavička `Accept-encoding`, ktorá prezrádza aké kompresné algoritmy prehliadač podporuje. Ďalej taktiež hlavička `Accept-language` prezrádzajúca, ktoré jazyky užívateľ

[1] Emscripten HTML5 API Eventy a funkcie orientácie zariadenia https://emscripten.org/docs/api_reference/html5.h.html#device-orientation

[2] Emscripten HTML5 API Eventy a funkcie batérie https://emscripten.org/docs/api_reference/html5.h.html#battery

[3] Emscripten HTML5 API Eventy a funkcie vibrácií. https://emscripten.org/docs/api_reference/html5.h.html#vibration

preferuje a hlavička `User-Agent` z ktorej vieme zistiť prehliadač, verziu prehliadača a verziu systému. Tieto dáta sa vytiahnu z GET požiadavky a pošlú sa naspäť na klientskú aplikáciu.

Vrátená odpoveď zo serveru bude obsahovať dáta, ktoré sa získali z GET požiadavky v serverovej časti. Tieto dáta potom zobrazíme naspäť užívateľovi a využijeme ich v celkovom dátovom hašovaní na jedinečnú identifikáciu zariadenia.

Metóda identifikácie reklámných blokerov

Kontrola dostupných blokerov reklám bude pracovať na princípe zaslania požiadavky. Z Web Assembly sa pošle GET požiadavka na reklamný server Googlu, ktorý sa nachádza na blokovacích listinách reklamných blokerov. Keď požiadavka nebude zablokovaná, môžeme predpokladať, že na prehliadači nie je dostupné, alebo je vypnuté rozšírenie na blokovanie reklám a takisto nie sú reklamy blokované na sieťovej úrovni programami, alebo sieťovými zariadeniami.

Pri zablokovanej požiadavke predpokladáme dostupnosť rozšírenia programu, alebo sieťového zariadenia slúžiaceho na blokovanie reklám.

Druhý typ testu zistí či je blokácia na dokumentovej úrovni. To nám prezradí či sa jedná o rozšírenie, program či o sieťové zariadenie na blokáciu reklám. Na zistenie dostupnosti rozšírenia sa vygeneruje HTML element, ktorému sa priradí trieda `ad` a identifikácia, pomocou ktorej budeme môcť pristupovať k testovaciemu reklamnému elementu. Rozšírenia blokujúce reklamné elementy pracujú na viacero spôsobov. Niektoré vymažú element obsahujúci triedy alebo identifikáciu reprezentujúcu reklamu, niektoré pridajú atribút `display: none;` reklamnému elementu, ktorý nám nezobrazí daný element na stránke a iné zmenia veľkosť reklamného elementu na hodnotu 0. Jedným z dostupných rozšírení, ktoré ma implementované odstraňovanie elementov zo stránky je rozšírenie UBlock, ktoré nám priamo umožňuje vybrať element na stránke, ktorý sa bude odstraňovať pri načítaní z DOM objektu.

Z týchto získaných informácii budeme schopní rozlíšiť medzi blokáciou reklám pomocou prehliadača alebo blokáciou pomocou programu či sieťového zariadenia. Prvý test, ktorý otestuje blokáciu požiadavky na reklamný server zistí či užívateľ využíva systém blokácie reklám. Druhý test nám otestuje či blokácia prebieha na úrovni prehliadača, alebo na úrovni ďalšieho programu, prípadne sieťového zariadenia.

Možný problém pri zasielaní GET požiadavky bude z SOP(Same-Origin-Policy)[4], ktorý bude blokovať požiadavky, ktoré sa budú zasielať na reklamné servery, pretože reklamný server nie je toho istého pôvodu, ako server na ktorom sa nachádza naša webová stránka na identifikáciu pomocou Web Assembly. Pretože reklamné blokery blokujú požiadavky na základe nájdenej zhody url v blokovacích listinách, stačí nám poslať požiadavku obsahujúcu reťazec dostupný v blokovacích listinách na server, ktorý hosťuje našu webovú stránku.

Nakoniec by som chcel ešte poznamenať, že identifikácia reklamných blokerov vyžaduje špeciálne potvrdenie od užívateľa, pretože sa jedná o ukladanie skriptov na užívateľovom zariadení. Potvrdenie je potrebné podľa štúdie [2], ktorá hovorí o tom, že identifikácia blokerov reklám je bez potvrdenia od užívateľa nelegálna. Preto by sa pred spustením identifikácie malo vyžiadať od užívateľa potvrdenie k vykonaniu identifikácie reklamných blokerov. Ak by užívateľ toto potvrdenie odmietol, mal by sa tento druh identifikácie preskakovať. My nebudeme implementovať tento typ kontroly, pretože sa jedná len o prácu v ktorej sa snažíme poukázať na možnosť identifikácie pomocou Web Assembly. A však pred spustením bude užívateľ upozornený, kde presne sa budú zasielať požiadavky.

[4]SOP https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

Metóda identifikácie zatemňovačov stránok

Na kontrolu dostupných zatemňovačov stránok nastavíme element body webovej stránky na bielu farbu a pridáme mu identifikáciu, pomocou ktorej k nemu budeme môcť pristupovať z web Assembly pomocou HTML5 API. Po načítaní stránky sa počká jednu sekundu, aby testovacie rozšírenie mohlo vykonať úpravu farby elementov. Po tejto sekunde sa skontroluje, či farba elementu body je stále rovnaká, aká bola preddefinovaná. Ak farba ostala nezmenená, môžeme predpokladať, že zatemňovač stránok nie je nainštalovaný alebo je vypnutý. Ak sa farba zmenila, existuje rozšírenie, alebo funkcionalita prehliadača, ktorý túto farbu zmenil.

4.3 Obmedzenia

Jedným z hlavných obmedzení je funkcionalita Web Assembly, ktorá musí byť na danom prehliadači zapnutá. Tor, Opera mini, Internet Explorer, UC Browser pre android, QQ Browser, Baidu Browser a KaiOS Browser nemajú podporu Web Assembly vôbec [14].

Ďalším obmedzením je Web Assembly, ktoré je ešte stále vo vývoji a obsahuje len základnú funkcionalitu. Jediný existujúci prístup k DOM štruktúre je prostredníctvom pár funkcií a eventov, ktoré využívajú JavaScript API.

Ďalším obmedzením je rýchlosť vývoja Web Assembly technológie v porovnaní s rýchlosťou vývoja webových štandardou, ktorým sa musí technológia prispôsobovať. Názorným príkladom je napríklad nový webový štandard virtuálnej reality WebXR. WebXR bol pridaný v prehliadači Chrome verzie 67 vydanej 29. Mája 2018[5], ako experimentálny štandard ktorý by zjednotil viaceré technológie virtuálnej reality do jedného API. 10 Decembra 2019 vyšla verzia prehliadača Chrome 79, v ktorej bola oficiálne odstránená podpora WebVR a WebXR sa stala oficiálnym štandardom webových technológii [10]. Po niekoľkých mesiacoch stále nie je aktualizovaná technológia Web Assembly na podporu novej WebXR API.

[5] História dátumov vydania prehliadača Google Chrome https://en.wikipedia.org/wiki/Google_Chrome_version_history

Kapitola 5

Implementácia

V tejto kapitole si vysvetlíme implementáciu programu, rozdelenie programu a vysvetlíme si ako sme implementovali viaceré metódy, ktoré sme si spomenuli v kapitole 4. Vysvetlíme si, jednotlivé zábrany v implementácii niektorých metód, ktoré nie sme schopní implementovať, pretože Web Assembly nám to neumožní. Na konci kapitoly bude tabuľka obsahujúca porovnanie hodnôt, ktoré môžme získať pomocou našej implementácie a webových stránok `AmIUnique`[1] a `Panopticlick`[2].

5.1 Implementácia programu

Program bude rozdelený na hlavnú a vedlajšiu časť, ako už bolo spomenuté v návrhu aplikácie (viď kapitola 4).

Hlavná časť bude implementovaná ako Web Assembly modul, ktorý bude využívať dostupné API na získanie informácii ohľadom užívateľského zariadenia. Modul bude spúšťaný z klienta po vyžiadaní od užívateľa klikom na tlačítko.

Vedlajšia časť bude implementovaná pomocou architektúry klient - server. Server (backend) bude slúžiť na obsluhu webovej stránky a na získanie http hlavičiek. Server bude implementovaný pomocou node frameworku Express[3], ktorý budem využívať hlavne preto lebo som s ním dobre oboznámený a je s ním veľmi jednoduché pracovať. Klient (frontend) bude obsahovať popis, kde užívateľovi vysvetlíme čo sa bude vykonávať, tabuľku, do ktorej zobrazíme užívateľovi získané informácie a tlačítko na spustenie identifikácie zariadenia. Po kliknutí na tlačítko užívateľom sa zavolá funkcia z Web Assembly modulu, ktorou sa spustí celý proces identifikácie zariadenia.

Zabezpečené spojenie

Spojenie medzi klientom a serverom je potrebné zabezpečiť, aby sme mohli využívať API na interakciu zo senzormi zariadení. Spojenie je zabezpečené pomocou HTTP serveru Nginx[4], ktorý zabezpečí spojenie pomocou SSL/TLS.

Aby sme sa vyhli upozorneniam o zabezpečení, potrebujeme certifikát, ktorý bude vydaný certifikačnou autoritou a ktorý bude dokazovať, že užívateľ komunikuje s tým, kto môže manipulovať s doménou. Certifikát je poskytovaný pomocou certifikačnej autority

[1] AmIUnique https://amiunique.org/
[2] Panopticlick https://panopticlick.eff.org/
[3] Express framework https://expressjs.com/
[4] Nginx https://www.nginx.com/

Lets Encrypt [5], ktorá zadarmo poskytuje certifikáty viacej než 200 milónom webových stránok.

Server (backend)

Ako bolo spomenuté v úvode tejto kapitoly, server využíva Express framework a pozostáva z jednoduchého API, ktoré odpovedá na GET požiadavky. Server hosťuje stránku a po zaslaní požiadavky na serverovú stránku `https://wasmfingerprint.me/get` navráti zoznam HTTP hlavičiek, ktoré obdržal zo zaslanej požiadavky. Tak isto po zaslaní požiadavky na serverovú url stránku `https://wasmfingerprint.me/get_ad/`, za ktorou môže nasledovať akýkoľvek reťazec, odpovie server prázdnym reťazcom. My budeme za serverovú url stránku, vkladať url reťazec reklamného serveru, čo sa bude využívať pri identifikácii blokerov reklám ako sme sa dozvedeli v časti 4.2.

5.1.1 Klient (frontend)

Klient je riešený pomocou jazykov HTML, CSS a Javascript. Po načítaní webovej stránky sa zobrazia užívateľovi informácie o webovej stránke, ktoré API sa budú využívať na identifikáciu zariadení a kde všade sa budú zasielať požiadavky. Výsledok identifikácie parametrov zariadenia bude zobrazený v prehľadnej html tabuľke. Po načítaní webovej stránky sa spustí proces inicializácie Web Assembly modulu, o ktorom sa dozvieme nižšie v podsekcií Inicializácia Web Assembly modulu.

Užívateľský vstup slúži nato, aby sme sa vyhli rôznym problémom z API, ktoré vyžadujú užívateľský vstup tak sa identifikácia zariadenia spustí až po kliknutí na tlačítko na webovej stránke. Tento spôsob bol implementovaný z toho dôvodu že napríklad audio API vyžaduje prvotný užívateľský vstup, na základe ktorého sa môže inicializovať audio kontext a dalej pracovať zo zvukom. Užívateľský vstup reprezentuje interakciu užívateľa zo stránkou, môže sa jednať napríklad o klik a potom všetky API, ktoré sa zavolajú z kliknutého tlačítka majú užívateľské povolenie na vykonanie svojej činnosti.

Inicializácia Web Assembly modulu prebieha pomocou Javascriptu. V Javascripte si vytvoríme pamäťový objekt (viď časť 3.1), z ktorým môže Web Assembly modul pracovať. K pamäťovému objektu môžme pristupovať ako z Web Assembly, tak isto aj z Javascriptu čo nám bude umožňovať predávanie dát. Ďalej je potrebné importnúť Javascriptové funkcie, pomocou ktorých budeme vyplňovať tabuľku dátami priamo z Web Assembly a exportnúť funkciu z Web Assembly modulu, pomocou ktorej budeme môcť z Javascriptu spustiť celý proces identifikácie zariadenia.

Import funkcií do WebAssembly je potrebný ak chceme mať aspoň čiastočnú interakciu z HTML DOM elementom. Z tohto dôvodu sme nútení vytvoriť funkciu, pomocou ktorej sme schopní vložiť dáta do tabuľky priamo z Web Assembly. Funkcia na vloženie dát do tabuľky obdrží identifikátor a dáta. Na základe obdržaného identifikátoru vieme kde presne do tabuľky vložiť obdržané dáta. Identifikátor spolu z dátami je potrebné zkonvertovať do správneho tvaru, kedže identifikátor a dáta obdržíme ako ukazateľ do vytvoreného pamäťo-

[5] Lets Encrypt https://letsencrypt.org/

vého objektu spolu z ich veľkosťami. Aby sme podporovali UTF-8 je potrebné konvertovať pole bytov do Javascript reťazca. čo je prevedené pomocou TextDecoder API[6].

Export funkcií z WebAssembly je potrebný hlavne preto, aby sme mohli zavolať funkciu mimo WebAssembly modulu. Exportujeme jedinú funkciu, ktorá nám spustí proces identifikácie parametrov zariadenia po stlačení tlačítka o ktorom sme sa dozvedeli v časti 5.1.1.

Prevzatá knihovňa, ktorú používam vo svojom projekte je CryptoJS. Knižnica má MIT licenciu a nachádza sa v zložke libraries spolu s licenčným súborom. Knižnica je dostupná zo služby Github na adrese `https://github.com/brix/crypto-js`. Využívam z tejto knižnice modul md5.js, ktorý obsahuje hašovaciu funkciu MD5, ktorou hašujem vyrenderovaný obrázok.

5.1.2 Web Assembly modul a implementované metódy identifikácie v module

Web Assembly modul ako názov udáva bude implementovaný pomocou Web Assembly. Kód bude písaný v jazyku C modulárnym štýlom programovania. Kód bude skompilovaný pomocou Emscripten knižnice (viď časť 3.2) do Web Assembly. Modul bude pozostávať z množstva metód na identifikáciu zariadení, ktoré si predstavíme v nasledujúcich sekciách.

Metóda identifikácia grafického hardwaru a rozpoznanie prehliadača Brave (verzia pre počítače)

Identifikácia grafického hardwaru spočíva vo vykreslení trojuholníka grafickou kartou. Vykreslený trojuholník je jedinečný pre model grafickej karty, čo znamená že keď dvaja užívateľia vlastnia identický model grafickej karty, výsledný vykreslený trojuholník bude dotožný.

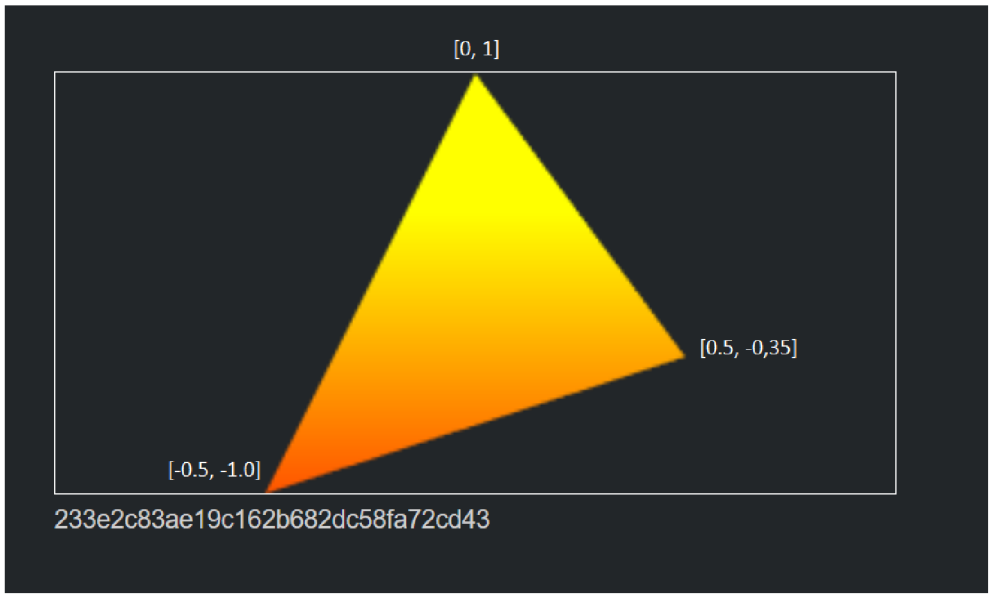
Súradnice trojuholníka, ako je možné vidieť na obrázku 5.1 boli špecificky zvolené s ohľadom na zvýšenie jedinečnosti vykreslenia. Jedinečnosť vykreslenia sa zvýši pomocou priamok medzi danými bodmi, ktoré vytvárajú rôzne stupne náklonu priamok. Je to hlavne preto, aby algoritmus na vykreslenie čiar, ktorý používa grafická karta mal viacero možností na vytvorenie odchyliek oproti ostatným algoritmom na vykreslenie čiar.

Po vyrenderovaní obrázka, máme obrázok stále načítaný vo vyrovnávacej pamäti, čo predáme hašovacej funkcii MD5, ktorá nám vygeneruje 128-bitový haš.

Vypnutá funkcionalita WebGL nám neumožňuje získať kontext grafického API, ktoré je potrebné na identifikáciu grafického hardwaru. Viacej je možné sa dozvedieť v časti 6.1, kde sa dozvieme o obmedzeniach danej metódy. Bohužial pri vypnutej funkcionalite WebGL nie sme schopní vykresliť trojuholník na získanie odtlačku grafickej karty. Vypnutá funkcionalita je možná iba v dvoch prípadoch. Prvý prípad značí zásah užívateľa ktorý túto funkcionalitu vypol z osobného dôvodu. Druhý prípad značí bota ktorý využíva headless prehliadač, ktorý je zbavený viacerých štandardných funkcionalít ako je napríklad WebGL.

Identifikácia prehliadača Brave je možná z toho dôvodu, že prehliadač má špecificky upravenú funkcionalitu WebGL. Brave nemá funkcionalitu WebGL vypnutú štandardným spôsobom, ale má ju upravenú tak, aby sa tvárila že funguje. Znamená to, že je možné

[6] TextDecoder API `https://developer.mozilla.org/en-US/docs/Web/API/TextDecoder`



Obrázek 5.1: Ukážka vykresleného trojuholníka pomocou OpenGL a MD5 hash danej ukážky

získať kontext WebGL, ale nie je možné s ním ďalej pracovať. Presnejšie postup vykreslenia trojuholníka zlyhá pri kompilácii vertex shaderu z neznámou chybou. Táto špecifická funkcionalita nám dovolí odhaliť prehliadač Brave. Funkcionalita je špecifická, ako pre počítačovú, tak isto aj pre mobilnú verziu prehliadača Brave.

Obmedzenie identifikácie grafického hardwaru, spočíva v možnosti úpravy WebGL API, ktoré slúži na vykreslovanie. API je možné upraviť pomocou rozšírenia, ktoré si spomenieme v časti 6.1. Identifikácia grafického hardwaru nie je nijako ovplyvnená metódami na zabránenenie identifikácie zariadenia, ktoré pracujú na Javascriptovej úrovni.

Metóda na odhad prehliadačov podľa dostupných HTTP hlavičiek

Odhad prehliadačov podľa dostupných HTTP hlavičiek je rozdelený na serverovú časť a klientskú časť. Serverová časť obdrží hlavičky požiadavky a uloží si tieto hlavičky do dočasného uložiska. Klientská časť zašle požiadavku a obdrží uložené http hlavičky zo servera. Obdržané http hlavičky o ktoré máme záujem, sú len tie, ktoré sme si spomenuli v časti 2.2. Obdržané http hlavičky zobrazíme užívatelovi v tabuľke, ktorú sme si spomenuli v časti 5.1.1. Na odhad prehliadačov skontrolujeme dostupnosť http hlavičiek typu Sec-Fetch(viac v časti 2.2, odstavec o hlavičke Sec-Fetch). Ak Sec-Fetch hlavičky sú súčasťou požiadavky, budeme predpokladať že sa jedná o prehliadače Brave, Google Chrome, Microsoft Edge alebo Operu. Ak požiadavka Sec-Fetch hlavičky neobsahuje predpokladáme, že sa jedná buď o Apple zariadenie, alebo prehliadač Firefox.

Metóda identifikácie dostupných mobilných senzorov

API na interakciu s mobilnými senzormi je jedno z tých API ktoré vyžadujú https na správne fungovanie, ako sme si spomenuli v časti 5.1. Na identifikáciu senzorov sa využíjú dve API. Prvé API nám umožní získať orientáciu zariadenia v priestore, API na orientáciu zariadenia)

a druhé API nám umožní získať smer ktorým sa zariadenie pohybuje. Obidve API sme si spomenuli v časti 3.2 v podsekciách API na orientáciu zariadenia a API na pohyb zariadenia. Identifikácia dostupných mobilných zariadení nedokáže zistiť či zariadenia obsahuje daný senzor alebo má iba k senzoru zakázaný prístup, lebo pri obidvoch možnostiach sa vracajú nulové hodnoty orientácie a pohybu.

Metóda identifikácie dostupných blokerov reklám

Identifikácia dostupných blokerov reklám pracuje na princípe zaslania požiadavky na reklamný server. Výsledok tejto požiadavky nám rozhodne o dostupnosti blokeru reklám. Viac sme si povedali o identifikácii blokerov reklám v časti 4.2. My nebudeme zasielať požiadavku na reklamný server ale zašleme požiadavku na server, ktorý hosťuje našu webovú stránku na identifikáciu zariadení aby sme sa vyhli SOP (Same-Origin-Policy). Nie je potrebné posielať požiadavku na reklamný server pretože reklamné blokery, kontrolujú url stránok a požiadaviek zo svojimi listinami reklamných stránok, čiže stačí aby súčasťou požiadavky bol reťazec ktorý bude zablokovaný na základe pravidla v blokačných listiných reklamných blokerov.

Metóda identifikácie rýchlosti kolieska myši a rozlíšenia displeja

Hodnota rýchlosti kolieska myši reprezentuje počet pixelov, o ktoré sa má stránka posunúť. Hodnotu získavame pomocou eventu kolieska myši, ktorý sa aktivuje pri pohybe kolieska myši. Táto hodnota je ovplyvňovaná výškou displeja, na ktorom je stránka zobrazená.

Rozlíšenie displeja sa počíta z hodnôt, výška a šírka displeja. Tieto hodnoty získame zo statusu fullscreen eventu, bez toho aby sa stránka musela prepnúť do fullscreen režimu. Získané hodnoty výšky a šírky displeja ešte nie sú kompletné a preto ich je potreba vynásobiť hodnotou pomeru pixelov zariadenia. Výsledkom vynásobených hodnôt bude výška a šírka displeja, čo je rozlíšenie displeja. Metóda identifikácie rozlíšenia displeja, môže byť ovplyvnená stupňom priblíženia webovej stránky, pretože rôzne prehliadače pracujú odlišnými spôsobmi pri približovaní webovej stránky.

Metóda identifikácie podpory dostupných funkcionalít webu

Prehliadače obsahujú viaceré funkcionality, ktoré môžu byť vypnuté rozšírením, alebo priamo užívateľom. My budeme kontrolovať Web Assembly, WebGL a IndexDB funkcionalitu. Spôsoby kontroly jednotlivých funkcionalít si vysvetlíme v ďalších podsekciách.

Podpora Web Assembly funkcionality sa kontroluje pomocou volania funkcie. Ak sa podarí spustiť prvotnú funkciu na identifikáciu, z ktorej sa volajú ostatné funkcie tak to znamená, že Web Assembly je podporované. Na druhú stranu ak sa nepodarí znamená to že funkcionalita Web Assembly nie je podporovaná.

Podpora WebGL funkcionality sa kontroluje pri inicializačnej fázy identifikácie grafického hardwaru, ktorú sme si spomenuli v časti 5.1.2. Inicializačná časť je implementovaná, ako pravdivostná funkcia, ktorá vráti pravdivostnú hodnotu true pokiaľ sa celá inicializačná časť podarila. Ak nastane chyba počas celého inicializačného procesu tak inicializačná funkcia vracia pravdivostnú hodnotu false. Na základe výsledku inicializačnej funkcie rozhodneme či WebGL funkcionalita je dostupná alebo nie.

Podpora IndexDB funkcionality je kontrolovaná cez jednoduché API, ktoré sme si spomenuli v časti 3.2. Pomocou IndexDB API uložíme reťazec slova `Hello world!` a následne po úspešnom uložení sa pokúsime získať reťazec, ktorý sme uložili. Ak počas týchto dvoch operácii nenastane žiadna chyba môžme predpokladať že IndexDB je podporovaná. Ak sa počas ukladania a načítania dát vyskytne chyba môžeme predpokladať, že funkcionalita IndexDB nie je dostupná.

5.2 Zábrany v implementácii identifikačných metód

Zábrany v implementácii identifikačných metód existujú hlavne z toho dôvodu, že Web Assembly je stále ešte len vo vývojovej fáze. Vývojovou fázou je myslené to že rôzne moduly na interakciu z webovými API ešte stále nie sú dokončené a ďalšie, ktoré fungujú, tak pracujú pomocou Javascriptu. Tak isto vývoj Web Assembly je vcelku zdĺhavý a pomalý, preto sa môže stať, že keď výjde nový webový štandard trvá aj niekoľko mesiacov, kým sa Web Assembly prispôsobí novému webovému štandardu. Preto je problémové implementovať niektoré identifikačné metódy, ktoré si spomenieme v nasledujúcich odsekoch.

Identifikácia pomocou audio kontextu nie je v súčasnej verzíí Web Assembly možná. Jedným z hlavných problémov je nedostupné API na interakciu z Audio API z Web Assembly. Je dostupná iba čiastočná podpora zvuku, ktorá pozostáva z nahrávania a prehrávania zvuku v 3D priestore. Bohužiaľ nie sme schopní implementovať ani jednu z metód identifikácie pomocou audio kontextu o ktorých sme sa dozvedeli v časti 2.1.2.

Identifikácia zatemňovačov stránok nie je v súčasnej verzií Web Assembly možná, pretože API na prístup k CSS hodnotám elementu, podporuje iba získanie šírky a výšky elementu a my vyžadujeme prístup k farbe elementu, aby bolo možné metódu identifikácie zatemňovačov stránky implementovať.

Identifikáciu počítača a mobilného zariadenia, je metóda, ktorú sme si predstavili v časti 4.2. Je to jedna z viacerých metód, ktorú nie sme schopní implementovať z toho dôvodu, že vyžaduje štyri API, z ktorých funguje iba API na detekciu pohybu a rotácie zariadenia. API ktoré vyžadujeme aby fungovali sú, API na interakciu z virtuálnou realitou, ktoré nefunguje vo Web Assembly ako sme si spomenuli v časti 3.2, API na zisťovanie stavu batérie a API na kontrolu vibrácií.

5.3 Porovnanie webových stránok na získanie informácií o zariadení

Porovnanie webových stránok určených na získanie informácií o zariadení a následnej identifikácii zariadenia medzi stránkami `panopticlick`[7] a `amiunique.org`[8] s mojou implementáciou stránky na získanie informácii pomocou Web Assembly je možné vidieť v tabuľke 5.1.

[7] Panopticlick https://panopticlick.eff.org/
[8] AmIUnique https://amiunique.org/

Detekované položky	AmIUnique	Panopticlick	Web Assembly Fingerprinting
Výpis pluginov	✓	✓	
Detekcia fontov	✓	✓	
HTTP Hlavička User-Agent	✓	✓	✓
HTTP Hlavička Accept	✓	✓	✓
HTTP Hlavička Accept-Encoding	✓		✓
HTTP Hlavička Accept-Language	✓		✓
HTTP Hlavička referer	✓		✓
HTTP Hlavička do not track	✓		✓
HTTP Hlavička Sec-Fetch-dest	✓		✓
HTTP Hlavička Sec-Fetch-site	✓		✓
HTTP Hlavička Sec-Fetch-mode	✓		✓
HTTP Hlavička Sec-Fetch-user	✓		✓
Odhad prehliadača na základe HTTP hlavičiek			✓
Rozlíšenie obrazovky	✓	✓	✓
Časová zóna	✓	✓	
Detekcia rýchlosti koliečka myši			✓
Jazyk prehliadača	✓		
Operačný systém a verzia kernelu	✓		
Zapnutý jazyk Java	✓		
Zapnuté cookies	✓	✓	
Dostupnosť WebAssembly			✓
Dátum a čas	✓		
Odtlačok canvasu	✓	✓	
Odtlačok WebGL	✓		✓
WebGL renderovací kontext	✓	✓	
Databáza indexedDB	✓		✓
Detekcia batérie	✓		
Odhad prehliadača Brave			✓

Tabulka 5.1: Tabulka reprezentuje, aké typy informácii, sú schopné webové stránky `AmIUnique` a `Panopticlick` zistiť pomocou JavaScriptu a porovnáva, aké informácie sme schopní získať pomocou našej implementácie stránky na získanie informácii vo Web Assembly. Zoznam testovacích hodnôt bol získaný z papieru [15] a doplnený o testovacie hodnoty z našej implementácie.

Kapitola 6

Testovanie a obmedzenia implementácie

V tejto kapitole sa budem zaoberať testovaním jednotlivých metód, ktoré sme implementovali v kapitole 5. Testovanie prebiehalo na viacerých zariadeniach (počítače, mobily s operačným systémom Android a Apple, MacBooky). Ďalej testovanie prebiehalo na prehliadačoch Opera, Brave, Google Chrome, Firefox, Safari a Microsoft Edge. Prehliadače Internet Explorer a Tor nie sú súčasťou testovania pretože obidva prehliadače nepodporujú Web Assembly. V nasledujúcich častiach si otestujeme viaceré metódy na identifikáciu zariadení, oproti niekoľkým rozšíreniam na zabránenie identifikácie zariadení.

6.1 Testovanie metódy identifikácie grafického hardwaru

Pri testovaní metódy na identifikáciu grafického hardwaru som našiel niekoľko obmedzení. Jedným z nich bolo, že WebGL funkcionalita bola vypnutá, čo nám zabránilo v získaní odtačku grafického hardwaru. Ďalším obmedzením je obmedzená funkcionalita pri prehliadači Brave, kde sa naskytne chybový stav pri kompilácii vertex shaderu, ktorý nám zabráni v pokračovaní vykresľovania. Táto chyba vedie k identifikácii prehliadača Brave, bola otestovaná 15. Apríla 2020 na verzii prehliadača Brave *1.7.92 Chromium: 80.0.3987.163 (Official Build) (64-bit)*.

Veľkým problémom metódy identifikácie grafického hardwaru je existencia rozšírenia **WebGL Fingerprint Defender**[1], ktorý upraví WebGL funkcionalitu prehliadača aby pridávala šum k vykreslenému obrázku. Pridaný šúm je dynamický a generuje sa náhodne pri každom novom pokuse o vykreslenie. Ďalším problémom tejto metódy je, ak zariadenie obsahuje viaceré grafické hardwery. Jednoduchým príkladom je notebook, ktorý obsahuje integrovanú grafickú kartu a dedikovanú grafickú kartu. Prehliadač môže využívať jednu z týchto grafických kariet a dokonca môže užívateľ zvoliť, ktorá grafická karta bude využitá na spracovanie grafických požiadaviek prehliadača.

Samozrejme metóda na identifikáciu grafického hardwaru pracuje správne, pri testovaní dvoch počítačov s identickým procesorom a grafickou kartou boli vygenerované haše grafických kariet identické.

Pri testovaní či sa haš zmení na rôznych prehliadačoch sme došli k záveru, že haše prehliadačov Google Chrome, Brave, Opera a Microsoft Edge sú identické, pokiaľ sa ne-

[1] Plugin WebGL Fingerprint Defender https://chrome.google.com/webstore/detail/webgl-fingerprint-defende/olnbjpaejebpnokblkepbphhembdicik

zmenili východzie nastavenia grafického hardwaru prehliača a zároveň testované zariadenia majú identickú dedikovanú grafickú kartu. Pokiaľ na počítači dedikovaná grafická karta chýbala, bolo potrebné aby testovacie zariadenia obsahovali identickú integrovanú grafickú kartu, pretože pri chýbajúcej dedikovanej grafickej karte, prehliadače využívali integrovanú grafickú kartu. Prehliadač Firefox má odlišný haš a to preto, lebo východzie nastavenia prehliadača Firefox využívajú integrovanú grafickú kartu oproti dedikovanej grafickej karte. Na zariadeniach, kde dedikovaná grafická karta chýbala mali všetky testované prehliadače identický haš, pretože všetky využívali integrovanú grafickú kartu.

6.2 Testovanie implementácie rozšírením Web API Manager v prehliadači Firefox

Web API Manager je rozšírenie pre prehliadač Firefox, ktoré umožňuje dočasne alebo natrvalo zakázať rôzne webové API. Pomocou tohto rozšírenia sme otestovali našu implementáciu identifikácie zariadení pomocou Web Assembly. Všetky testy boli vykonané ako na počítačovej verzii prehliadača Firefox tak isto aj na mobilnej verzii prehliadača Firefox. Všetky testy mali identické výsledky, ako na počítačovej verzii, tak isto aj na mobilnej verzii prehliadača Firefox.

Prvý test pozostával z vypnutia IndexDB API, ktorý umožňuje prístup k databáze do ktorej sme schopní uložiť dáta. Naša implementácia, ktorú sme si predstavili v kapitole 5, úspešne identifikuje nedostupnú podporu IndexDB API. Pri zapnutom IndexDB API tak isto úspešne identifikuje dostupnú podporu API.

Druhý test pozostával z vypnutia HTML Canvas elementu, čo malo za následok zlyhanie pri kompilácii vertex shaderu, ktorý je súčasťou identifikácie grafického hardwaru ako sme si spomenuly v časti 5.1.2. Zlyhanie nastalo presne v tej istej časti kódu, ktorú využívame na identifikáciu prehliadača Brave, ktorú sme si tak isto spomenuli v časti 5.1.2. To isté zlyhanie v časti kódu, nastane aj vtedy keď vypneme WebGL Specification API. Rozdiel v zlyhaní pri vypnutom HTML Canvas elemente spolu s WebGL Specification API a prehliadačom Brave je v navrátenej chybovej hodnote, ktorá sa líši pri prehliadači Brave a vypnutých už spomínaných API využitých pri tomto teste. Zlyhanie neni neočakávané nakoľko proces renderácie vyžaduje kontext WebGL, ktorý sa musí získať z dostupného HTML Canvas elementu.

Tretí test spočíval v zakázaní XMLHttpRequest API, ktoré slúži na zasielanie požiadaviek priamo z Web Assembly kódu. Ako sme predpokladali, všetky metódy na získanie informácii o zariadení, ktoré využívajú Fetch API (viz. časť 3.2 Fetch API), ktoré využíva XMLHttpRequest API sú nefunkčné. Jedná sa hlavne o metódu identifikácie HTTP hlavičiek a následovný odhad prehliadača na základe získaných hlavičiek.

Štvrtý test pozostával zo zakázania funkcionalít jadra webovej stránky, kde sme zakázali DOM Level 2 eventy spolu z encoding funkcionalitov. Vypnutie týchto dvoch funkcionalít nám zabránilo v získaní rozlíšenia displeja a v získaní hodnoty rýchlosti kolieska myši, ktorá reprezentuje počet pixelov, o ktoré sa stránka musí posunúť.

Keď sme zakázali všetky funkcionality webovej stránky, ktoré sa postupne zakazovali počas testov, tak sme boli schopní úplne odstaviť našu implementáciu identifikácie zariadení od identifikovania zariadenia.

6.3 Testovanie metód identifikácie rýchlosti koliečka myši a rozlíšenia displeja

Implementácia metódy na identifikáciu rýchlosti kolieska myši a rozlíšenie displeja bola rozobratá v časti 5.1.2.

Hodnota rýchlosti koliečka myši, reprezentujúca počet pixelov, o ktoré sa má posunúť webová stránka, bola identická pri prehliadačoch Brave, Google Chrome a Opera. Prehliadač Microsoft Edge mal odlišnú hodnotu od testovaných prehliadačov. Test pri prehliadači Firefox vracal statickú hodnotu 3.0.

Rozlíšenie displeja bolo testované na viacerých zariadeniach z rôznymi rozlíšeniami. Rozlíšenie displeja bolo identické pri testovacej skupine prehliadačov Firefox, Brave, Google Chrome, Opera, Microsoft Edge. Rozlíšenia displejov odpovedali reálnemu rozlíšeniu displejov. Problém však nastal keď sa dokument približoval a odďaľoval, vtedy rozlíšenie displeja neodpovedalo reálnemu rozlíšeniu displeja, ako sme si už spomenuli v časti 5.1.2.

6.4 Testovanie metódy identifikácie blokerov reklám

Zisťovanie blokerov reklám bolo otestované na prehliadačoch Google Chrome, Firefox, Brave, Opera a Microsoft Edge. Testované rozšírenia boli uBlock Origin[2], AD Block[3], AD Block Plus[4], Ghostery[5], AdGuard[6]. Testovanie prebiehalo tak, že sa postupne na každé zariadenie nainštalovalo jedno z testovaných rozšírení a overilo sa či webová stránka identifikovala správne dostupný bloker reklám. Test splnil naše očakávania, webová stránka úspešne identifikovala dostupné testované rozšírenia na blokovanie reklám dokonca bola schopná identifikovať východzí bloker reklám dostupný v prehliadači Opera, avšak nedokázala identifikovať bloker reklám implementovaný v prehliadači Brave. Je to hlavne z toho dôvodu že prehliadač Brave neblokuje požiadavky zasielané na reklamné servery. Keď prehliadač Brave obsahoval rozšírenie na blokovanie reklám, to sme už boli schopní identifikovať.

6.5 Testovanie implementácie rozšírením JS Restrictor

Rozšírenie JS Restrictor, ktoré sme si spomenuli v časti 2.4.1, bolo otestované na troch prehliadačoch Google Chrome, Firefox a Opera. Nebolo možné ho otestovať na prehliadači Microsoft Edge pretože rozšírenie neexistuje pre tento prehliadač. Výsledok testu bol veľmi zaujímavý a to hlavne preto lebo JS Restrictor bol schopný upraviť hlavičku User Agent (viď časť 2.2, odstavec o hlavičke User-Agent) spolu z hlavičkou Accept-Language (viď časť 2.2, odstavec o hlavičke Accept-Language) a tak isto aj hlavičku Referer(viď časť 2.2, odstavec o hlavičke Referer). Tak isto aj XMLHttpRequest API pri zaslaní požiadavky na server hosťujúci webovú stránku upozorní užívateľa na zaslanú požiadavku a umožní

[2]Rozšírenie uBlock Origin https://chrome.google.com/webstore/detail/ublock-origin/cjpalhdlnbpafiamejdnhcphjbkeiagm?hl=sk

[3]Rozšírenie AD Block https://chrome.google.com/webstore/detail/adblock-%E2%80%94-best-ad-blocker/gighmmpiobklfepjocnamgkkbiglidom

[4]Rozšírenie AD Block Plus https://chrome.google.com/webstore/detail/adblock-plus-free-ad-bloc/cfhdojbkjhnklbpkdaibdccddilifddb

[5]Rozšírenie Ghostery https://chrome.google.com/webstore/detail/ghostery-%E2%80%93-privacy-ad-blo/mlomiejdfkolichcflejclcbmpeaniij

[6]Rozšírenie AdGuard https://chrome.google.com/webstore/detail/adguard-adblocker/bgnkhhnnamicmpeenaelnjfhikgbkllg

mu ju zablokovať. Ďalej JS Restrictor obsahuje ochranu proti identifikácii pomocou canvas elementu, no nám to vôbec neovplyvní náš výsledný haš grafického hardwaru.

6.6 Testovanie prístupu Web Assembly funkcií k vstavaným objektom

Rozšírenie JS Restrictor, bolo takisto využité aj pri testovaní prístupu Web Assembly funkcií k vstavaným objektom. Na testovanie boli vybrané dve funkcie, ktoré sme schopní otestovať pomocou rozšírenia JS Restrictor. Prvá funkcia je funkcia, ktorá nám z Date objektu, vráti čas v milisekundách, ktorý uplynul od 1. Januára 1970. Druhá funkcia nám vráti presný čas v milisekundách, ktorý uplynul od načítania webovej stránky, ktorá využíva funkciu `performance.now()`. Test prebiehal následovne, zapamätali sme si hodnoty vrátené prvou a druhou funkciou a vypísali sme do konzoly správu. Využíli sme Web Assembly spánkovú funkciu, kde sme uspali program na 500 milisekúnd. Po prebudení, sme vypísali do konzoly správu, aby sme otestovali či program naozaj spal potrebný čas. Ďalej sa znova zavolala prvá a druhá funkcia a urobil sa rozdiel, medzi výsledkom volania prvej funkcie a uloženou hodnotou prvej funkcie. Ten istý rozdiel sa urobil aj pre druhú funkciu a vypísali sa rozdiely týchto dvoch volaní. Hodnoty rozdielov, ktoré sme získaly bez ovplyvnenia rozšírením JS Restrictor sa pohybovali okolo hodnoty 500, pretože spánková funkcia neni sama o sebe presná. Znova sa spustil ten istý test s tým rozdielom, že sme zapli rozšírenie JS Restrictor, v ktorom sme nastavili zaokrúhľovanie času objektu Date spolu zo zaokrúhľovaním času funkcie `performance.now()` na sekundy. Získané hodnoty rozdielov, boli obidve ovplyvnené rozšírením JS Restrictor, pretože obidve vrátili hodnotu 0. Výsledok tohto testu nám preukázal že Web Assembly nepristupuje k vstavaným objektom, ale pristupuje k objektom, ktoré sme schopní upraviť z Javascriptu.

Kapitola 7

Záver

V práci sme sa zoznámili z rôznymi metódami na identifikáciu informácii o zariadení, ktoré sme využili v implementačnej časti. Tak isto sme sa dozvedeli aj o metódach, rozšíreniach a prehliadačoch zameraných na zabránenie získaniu informácii o zariadení, ktoré by mohli viesť k identifikácii zariadení. Zoznámili sme sa s prehliadačom Brave, ako zabraňuje rôznym metódam identifikácie a prišli sme aj nato, ako rozoznať prehliadač Brave od ostatných prehliadačov. Predstavili sme si technológiu Web Assembly, pomocou ktorej sme implementovali metódy na identifikáciu informácii o zariadení, v podobe Web Assembly modulu. Tento modul je umiestnený na našej webovej stránke, kde pracuje a identifikuje zariadenia. Našu webovú stránku sme využívali na testovanie viacerých zariadení a metód na zabránenie identifikácie zariadení.

Implementácia identifikácie zariadení vo Web Assembly, je viacej obmedzovaná oproti implementácii v Javascripte a preto by som Web Assembly využil, buď v kombinácii z Javascriptom ako bol navrhnutý, čo sme si spomenuly v kapitole 3, alebo radšej využíval implementáciu v Javascripte. Po dokončení implementácie a otestovaní, som došiel k záveru, že identifikáciu zariadení pomocou Web Assembly je možné úplne zablokovať pomocou rozšírenia Web API Manager, ktoré zablokuje všetky API, ku ktorým pristupujeme a tým nám zabráni v získaní identifikácie zariadenia. V jednom z našich viacerých testov sme otestovali, či Web Assembly pristupuje k vstavaným objektom alebo k objektom, ktoré sú prístupné z Javascriptu. Výsledkom testu (viď časť 6.6) sme zistili že Web Assembly pristupuje k objektom, ktoré sú pristupné z Javascriptu a tým podliehajú možnej úprave priamo z Javascriptu, čo naznačuje aj test z rozšírením Web API Manager (viď časť 6.2). Ako sme si v teste na prístup k objektom ukázali, je možné testované časové funkcie upraviť pomocou rozšírenia JS Restrictor, aby sa presnosť časových funkcii znížila na sekundy.

Web Assembly sa z môjho pohľadu ešte stále nachádza v alpha verzii a preto každou novou aktualizáciou sa pridá nová funkcionalita, ale zároveň niečo čo už pracovalo správne prestane fungovať. Práve táto skutočnosť je jeden z veľkých dôvodov prečo ešte Web Assembly, po troch rokoch existencie neni až tak rozšírený. Práve preto by som s pokračovaním v tejto práci osobne počkal, kým sa technológia Web Assembly nedostane do bodu, kedy sa bude nachádzať v stabilnej verzii a bude mať prístup k značnému počtu webových API.

Literatura

[1] *Canvas Fingerprinting.* Navštívené: 10. Januára 2020. Dostupné z: https://browserleaks.com/canvas.

[2] *Directive 2002/58/EC of the European Parliament and of the Council of 12 July 2002.* July 2002. Navštívené: 10. Januára 2020. Dostupné z: https://eur-lex.europa.eu/eli/dir/2002/58/oj.

[3] *AudioContext Fingerprinting.* Sep 2017. Navštívené: 26. Apríla 2020. Dostupné z: https://www.darkwavetech.com/index.php/device-fingerprint-blog/audiocontext-fingerprinting.

[4] Bondy, B. R. *Brave Fingerprinting Protection Mode* [online]. Navštívené: 10. Januára 2020. Dostupné z: https://github.com/brave/brave-browser/wiki/Fingerprinting-Protection-Mode.

[5] Clark, L. *Memory in WebAssembly (and why it's safer than you think).* Jul 2017. Navštívené: 10. Januára 2020. Dostupné z: https://hacks.mozilla.org/2017/07/memory-in-webassembly-and-why-its-safer-than-you-think/.

[6] Copes, F. *An introduction to WebAssembly.* Nov 2018. Navštívené: 10. Januára 2020. Dostupné z: https://flaviocopes.com/webassembly/.

[7] Emam, K. a Alvarez, C. A critical appraisal of the Article 29 Working Party Opinion 05/2014 on data anonymization techniques. *International Data Privacy Law.* Leden 2014, roč. 5, s. 73–87. Navštívené: 10. Januára 2020.

[8] *Funkcia prehliadača: performance.now().* Navštívené: 10. Januára 2020. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Performance/now.

[9] Guttmann, A. *Ad blocking usage.* Aug 2019. Navštívené: 16. Apríla 2020. Dostupné z: https://www.statista.com/statistics/804008/ad-blocking-reach-usage-us/.

[10] Hoffman, C. *What's New in Chrome 79, Available Now.* How-To Geek, Dec 2019. Navštívené: 15. Apríla 2020. Dostupné z: https://www.howtogeek.com/450613/whats-new-in-chrome-79-arriving-today/.

[11] Laperdrix, P., Bielova, N., Baudry, B. a Avoine, G. Browser Fingerprinting: A survey. *CoRR.* 2019, abs/1905.01051. Dostupné z: http://arxiv.org/abs/1905.01051.

[12] Lardinois, F. *Google proposes new privacy and anti-fingerprinting controls for the web.* TechCrunch, Aug 2019. Navštívené: 10. Januára 2020. Dostupné z: https://techcrunch.com/2019/08/22/google-proposes-new-privacy-and-anti-fingerprinting-controls-for-the-web/.

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

[14] *Support tables for Web Assembly* [online]. Navštívené: 10. Januára 2020. Dostupné z: https://caniuse.com/#feat=wasm.

[15] Torres, C. F., Jonker, H. a Mauw, S. FP-Block: Usable Web Privacy by Controlling Browser Fingerprinting. In: Pernul, G., Y A Ryan, P. a Weippl, E., ed. *Computer Security – ESORICS 2015*. Cham: Springer International Publishing, 2015. ISBN 978-3-319-24177-7.

[16] Vastel, A., Laperdrix, P., Rudametkin, W. a Rouvoy, R. FP-scanner: The Privacy Implications of Browser Fingerprint Inconsistencies. In: *Proceedings of the 27th USENIX Conference on Security Symposium*. Berkeley, CA, USA: USENIX Association, 2018, s. 135–150. SEC'18. ISBN 978-1-931971-46-1.

[17] Welles, R. *Noscript Chrome Noscript Firefox – Browser Fingerprint Protection* [online]. 2019 [cit. !26-12-2019]. Dostupné z: http://www.idcloak.com/learning-center/noscript-chrome-noscript-firefox-browser-fingerprint-protection/a583.html.