VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ZÍSKÁVÁNÍ INFORMACÍ O UŽIVATELÍCH NA WEBOVÝCH STRÁNKÁCH

THESIS TITLE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. TOMÁŠ VONDRÁČEK**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Vondráček Tomáš, Bc.**
Program: Informační technologie a umělá inteligence Specializace: Vývoj aplikací
Název: **Získávání informací o uživatelích na webových stránkách**
Browser and User Fingerprinting for Practical Deployment
Kategorie: Web

Zadání:

1. Nastudujte možnosti jazyka JavaScript a dalších webových technologií v oblasti získávání otisku zařízení a dalších informací o uživatelích jako je analýza využívání vstupních zařízení (myš, klávesnice apod.).
2. Najděte webová rozšíření bránící webovým stránkám v získávání otisků technikami zjištěnými v bodě 1, popište jejich výhody a nevýhody.
3. Proveďte analýzu dostupných knihoven využívající techniky nastudované v bodě 1 zadání. Uveďte do jaké míry ovlivňují kvalitu otisků vytvářených analyzovanými knihovnami rozšíření nalezená v bodě 2 zadání.
4. Navrhněte vlastní nástroj (např. knihovnu v jazyce JavaScript) pro získávání informací o uživatelích, jejich prohlížečích a zařízeních.
5. Návrh implementujte.
6. Implementaci otestujte. Vyhodnoťte dopady využití rozšíření nalezených v bodě 2 zadání na kvalitu dat získávaných implementovaným nástrojem.
7. Práci vyhodnoťte a navrhněte možná zlepšení.

Literatura:

- Pierre Laperdrix, Nataliia Bielova, Benoit Baudry a Gildas Avoine. 2020.Browser fingerprinting: A survey. ACM Trans. Web, roč. 14, č. 2, str. 8:1-8:33.
- Antoine Vastel, Pierre Laperdrix, Walter Rudametkin a Romain Rouvoy. 2018. FP-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies. Proceedings of the 27th USENIX Security Symposium. Baltimore, United States.
- Umar Iqbal, Steven Englehardt, Zubair Shafiq. 2021. Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors. IEEE Symposium on Security & Privacy (S&P). Pre-print a další data dostupné online, URL https://uiowa-irl.github.io/FP-Inspector/.
- WP29 - Article 29 Data Protection Working Party. 2014 Opinion 9/2014 on the application of Directive 2002/58/EC to device fingerprinting. WP224, dostupné online https://ec.europa.eu/justice/article-29/documentation/opinion-recommendation/files/2014/wp224_en.pdf.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání, rozpracovaný bod 4.

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

Vedoucí práce: **Polčák Libor, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 26. října 2020

Abstrakt

Cílem diplomové práce je zmapovat informace poskytované webovými prohlížeči, které mohou být v praxi použity k identifikaci uživatelů na webových stránkách. Práce se zaměřuje na získání a následnou analýzu informací o zařízeních, prohlížečích a vedlejších efektů způsobených webovými rozšířeními, které maskují identitu uživatelů. Získání informací realizuje navržená a implementovaná knihovna v jazyce TypeScript, která byla nasazena na 4 komerčních webových stránkách. Analýza získaných informací je realizována po měsíci provozu knihovny a zaměřuje se na míru získané informace, rychlost získání informací a stabilitu informací. Z datové sady vyplývá, že až 94 % potenciálně různých uživatelů disponuje unikátní kombinací informací. Hlavní přínos této práce spočívá ve vytvořené knihovně, návrhu nových metod získávání informací, optimalizace stávajících metod a určení kvalitních a nekvalitních informací na základě jejich míry informace, rychlosti získání a stability v čase.

Abstract

The aim of the diploma thesis is to map the information provided by web browsers, which can be used in practice to identify users on websites. The work focuses on obtaining and subsequent analysis of information about devices, browsers and side effects caused by web extensions that mask the identity of users. The acquisition of information is realized by a designed and implemented library in the TypeScript language, which was deployed on 4 commercial websites. The analysis of the obtained information is carried out after a month of operation of the library and focuses on the degree of information obtained, the speed of obtaining information and the stability of information. The dataset shows that up to 94 % of potentially different users have a unique combination of information. The main contribution of this work lies in the created library, design of new methods of obtaining information, optimization of existing methods and the determination of quality and poor quality information based on their level of information, speed of acquisition and stability over time.

Klíčová slova

digitální otisk, otisk prohlížeče, otisk uživatele, otisk zařízení, identifikace uživatele, sledování uživatele, webový prohlížeč, entropie, stabilita, nekonzistence, anomálie, webové rozšíření, JavaScript, TypeScript, Cypress, Firebase, REST API

Keywords

digital fingerprint, browser fingerprint, user fingerprint, device fingerprint, user identification, user tracking, web browser, entropy, stability, inconsistency, anomaly, web extensions, JavaScript, TypeScript, Cypress, Firebase, REST API

Citace

VONDRÁČEK, Tomáš. *Získávání informací o uživatelích na webových stránkách.* Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Libor Polčák, Ph.D.

Získávání informací o uživatelích na webových stránkách

Prohlášení

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

.....
Tomáš Vondráček
13. května 2021

Poděkování

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Liborovi Polčákovi, Ph.D. za příkladné vedení práce, vstřícnost a čas věnovaný na konzultacích.

Obsah

Kapitola 1

Úvod

V minulosti byly velmi často pro účely identifikace uživatelů zneužívány soubory cookies, ve kterých je možné uložit identifikátor uživatele. Nevýhoda této metody spočívá v omezené době životnosti souborů cookies a možnosti uživatele kdykoliv soubory cookies smazat. Na základě těchto nedostatků se zrodila technika identifikace uživatelů pomocí získávání informací o zařízení a prohlížeči. Informace o zařízení a prohlížeči nelze ze své podstaty nikterak smazat, ale na druhou stranu mohou být ovlivněny například aktualizací operačního systému nebo prohlížeče. S rostoucí popularitou této nové techniky již vznikly desítky webových rozšíření, které maskují informace použitelné pro identifikaci a které si uživatelé mohou instalovat do svých prohlížečů a zvýšit tak ochranu své identity.

Cílem práce je vytvořit knihovnu, jejíž zodpovědností je získat maximálně autentická data o uživatelově zařízení, prohlížeči a jeho interakci s webovou stránkou. Důraz je taktéž kladen na kvalitu kódu, udržovatelnost a zajištění funkcionality i ve starších prohlížečích. Mimo knihovnu je implementováno webové rozšíření, které je určeno pro dlouhodobé získávání informaci o uživatelích, aby bylo možné lépe vyhodnotit stabilitu informací v čase. Jelikož je knihovna určena pro nasazení v praxi, je implementován i server, který zpřístupňuje kód knihovny a následně ukládá přijatá data z knihovny do databáze.

Kapitola 2 popisuje prerekvizity, na jejichž znalost je navázáno v následujících kapitolách. Následující tři kapitoly se věnují odlišným druhům informací, které lze získat z webového prohlížeče. Kapitola 3 popisuje druhy informací, které lze získat bezprostředně poté, co uživatel navštíví webovou stránku. Jako takové jsou velmi praktické a je jim věnována největší pozornost z celé práce. Kapitola 4 popisuje druhy informací, které lze získat na základě uživatelovy interakce s webovou stránkou. Interakce může probíhat například pomocí myši, klávesnice nebo dotyku. V následující kapitole 5 jsou rozebrány a otestovány techniky, kterými lze detekovat přítomnost webových rozšíření maskujících identitu uživatele. Kapitola 6 analyzuje knihovny pro získávání informací na základě znalostí z předešlých kapitol. Kapitoly 7 a 8 se věnují návrhu a implementaci systému pro získávání informací, který se skládá z knihovny, webového rozšíření, serveru a databáze. Kapitola 9 se zabývá analýzou dat, která byla získávána po dobu 1 měsíce na 4 komerčních webových stránkách prostřednictvím implementovaného systému.

Kapitola 2

Prerekvizity

Cílem této kapitoly je obeznámit čtenáře s důležitými a specifickými vlastnostmi programovacího jazyka JavaScript. Na informace z této kapitoly bude navázáno v některých z následujících kapitol.

2.1 Jazyk JavaScript a jeho důležité vlastnosti

Jazyk JavaScript je nejrozšířenější programovací jazyk používaný pro implementaci uživatelských rozhraní na webových stránkách. Jako takový má přístup k velkému množství webových rozhraní, ze kterých lze získat mnoho informací o prohlížeči, zařízení či uživateli.

2.1.1 Interní reprezentace objektů

Každý objekt, funkce je také objekt, obsahuje interní metody, které definují sémantiku objektu, a interní sloty (dále jen sloty), které definují interní stav objektu [19]. Interní metody a sloty jsou ve specifikaci značeny jménem ve dvojitých hranatých závorkách `[[]]`. Například veškeré funkce obsahují interní metodu `[[Call]]` a jako takové jdou zavolat. Sloty se na objektu vytvoří automaticky při jeho vytvoření a nelze je dynamicky přidávat.

2.1.2 Dědičnost v jazyce JavaScript

Jazyk JavaScript využívá koncept prototypové dědičnosti [19], která je realizována pomocí slotu `[[Prototype]]`. Hodnota slotu na objektu je buď `null`, nebo obsahuje referenci na další objekt. Důsledek prototypové dědičnosti je takový, že pokud daný objekt neobsahuje hledaný atribut, deleguje tento dotaz na svůj prototyp. Hodnota ze slotu lze získat zavoláním funkce `Object.getPrototypeOf`[1], nebo pomocí dnes již zastaralého, nicméně stále funkčního atributu `__proto__`[2].

2.1.3 Dotazování na vlastnosti média

Dotazování na vlastnosti média (angl. Media Queries) [48] je způsob, jak získat informace o zobrazovacím zařízení, na kterém je právě zobrazována webová stránka. Pro stolní počítač je typickým zobrazovacím zařízením monitor a pro mobilní zařízení zase jejich displej.

[1] developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getPrototypeOf

[2] developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/proto

Dotazování na vlastnosti média bylo nejdříve možné pouze v jazyce CSS, a to od 3. verze. Specifikace CSSOM View Module [47] definuje funkci `matchMedia`, která umožňuje dotazování na vlastnosti média přímo z jazyka JavaScript.

2.2 Jádro webových prohlížečů

Webový prohlížeč (dále jen prohlížeč) obsahuje jádro, které se stará o jeho fungování. Pro tuto práci je však důležitý fakt, že samotné jádro se skládá ze dvou částí, a to jádra pro vykreslování obsahu a jádra pro vykonávání kódu v jazyce JavaScript. [36]

2.2.1 Jádro pro vykreslování obsahu

Jádro pro vykreslování obsahu (angl. rendering engine) má na starost vizuální podobu stránky. V následujícím seznamu jsou asociována vykreslovací jádra použitá v konkrétních rodinách prohlížečů.

- **WebKit** (vychází z jádra KHTML) – Safari
- **Blink** (vychází z jádra WebKit) – Prohlížeče založené na projektu Chromium. Do této rodiny prohlížečů patří: Google Chrome, Seznam.cz, Brave, Opera (od verze 15), Edge (od verze 79).
- **Gecko** – Mozilla Firefox, Tor Browser (vychází z prohlížeče Mozilla Firefox)
- **Zastaralá vykreslovací jádra:**
  - **Presto** – Opera
  - **Trident** – Internet Explorer
  - **EdgeHTML** (vychází z jádra Trident) – Edge

2.2.2 Jádro pro vykonávání kódu v jazyce JavaScript

Jádro pro vykonávání kódu v jazyce JavaScript (angl. JavaScript engine) provádí a optimalizuje kód v jazyce JavaScript. V následujícím seznamu jsou asociována jádra pro vykonávání jazyka JavaScript s kompatibilními jádry pro vykreslování obsahu.

- **JavaScriptCore** (též známé jako Nitro) – WebKit
- **V8** (používané samostatně i v Node.js) – Blink
- **SpiderMonkey** – Gecko
- **Chakra** – Trident a EdgeHTML

2.2.3 Závěr

Rozdílná jádra mezi rodinami prohlížečů jsou důvodem toho, že je nutné vždy webové aplikace testovat napříč rodinami prohlížečů, protože jádra nemusí vůbec podporovat určitou funkcionalitu, nebo ji mohou realizovat s mírnými rozdíly. Naopak lze očekávat, že prohlížeče v rámci jedné rodiny prohlížečů budou vykazovat velmi podobné chování.

V následujících kapitolách bude k prohlížečům majícím stejná jádra referováno jako k rodině prohlížečů. Pokud bude například uvedeno, že určitá funkcionalita je podporována v rodině prohlížečů Blink, jsou tím myšleny prohlížeče založené na projektu Chromium. Rodina prohlížečů WebKit obsahuje pouze prohlížeč Safari.

Kapitola 3

Získávání informací o zařízení a prohlížeči

Kapitola popisuje informace, které lze získat z prohlížeče bezprostředně po načtení webové stránky. Informace jsou velmi detailně popsány, aby bylo možné pochopit jejich význam a jejich případné vazby na jiné informace. Informace jsou v kapitole členěny podle rozhraní, ze kterých je lze získat, nebo podle způsobu jejich získání.

3.1 Objekt navigator

Objekt `navigator` [54, 40, 1] je atribut na rozhraní `window` reprezentující stav a identitu prohlížeče. Atributy na objektu `navigator` obsahují informace o prohlížeči, nastavení systému a hardwarových dispozicích daného zařízení. Specifikace objektu `navigator` neudává přesné pořadí, v jakém by při iteraci měly být dostupné jeho atributy. Prohlížeče také někdy implementují experimentální nebo nestandardní funkcionality. Tyto funkcionality mívají prefix[1] dle daného typu prohlížeče např. `webkit` pro Chrome, Safari a Operu či `moz` pro Firefox.

V kapitole budou představeny především standardní a nezastaralé atributy objektu `navigator`, ze kterých lze získat informace. V případě, že je nějaký atribut experimentální či zastaralý, bude u něho tato informace explicitně uvedena.

3.1.1 User Agent

Atribut `userAgent` [1] obsahuje informace o prohlížeči a zařízení ve formátu řetězce. Řetězec musí obsahovat stejnou hodnotu jako hlavička `User-Agent` z protokolu HTTP. Syntaxe řetězce je uvedena ve specifikaci protokolu HTTP [39, 14], avšak specifikace je příliš vágní a prakticky dovoluje prohlížečům implementovat řetězec dle vlastního uvážení.

V roce 1993 používal první prohlížeč Mosaic velice jednoduchý formát řetězce ve formě `Mosaic/0.9`. Od té doby se formát řetězce již mnohokrát změnil. V době psaní této práce se formáty řetězců liší napříč prohlížeči. Mezi časté formáty řetězců se řadí následující dva, ve kterých jsou podtržením znázorněny informace, které se mění v závislosti na prohlížeči nebo zařízení. [16]

[1] https://developer.mozilla.org/en-US/docs/Glossary/Vendor_Prefix

Prohlížeče z rodiny Gecko.

```
Mozilla/MozillaVersion
(Platform; Encryption; OS-or-CPU; Language; PrereleaseVersion)
Gecko/GeckoVersion ApplicationProduct/ApplicationProductVersion
```

Prohlížeče z rodiny WebKit na mobilním zařízení.

```
Mozilla/5.0 (Platform; Encryption; OS-or-CPU like Mac OS X; Language)
AppleWebKit/AppleWebKitVersion (KHTML, like Gecko) Version/BrowserVersion
Mobile/MobileVersion Safari/SafariVersion
```

Z těchto různých formátů řetězců lze získat mnoho rozličných informací, jako například:

- **typ webového prohlížeče** (včetně verze) – Chrome, Firefox, Safari, ...
- **jádro pro vykreslování obsahu** (včetně verze) – Blink, Gecko, WebKit, ...
- **operační systém** (včetně verze) – Windows, Linux, iOS, Android, ...
- **architekturu procesoru** – amd64, arm[64], 68k, ...
- **typ zařízení** (včetně prodejce) – mobil, tablet, chytrá televize, ...

3.1.2 Platforma

Atribut `platform` [1] je řetězec reprezentující platformu, na které běží prohlížeč. Hodnota platformy může být i prázdný řetězec. Uvedená hodnota platformy by měla odpovídat typu operačního systému z řetězce `userAgent`. S hodnotami to však není vždy jednoznačné, protože mohou odpovídat například typu zařízení (iPhone, iPad), nebo verzi operačního systému (FreeBSD, Win32). V praxi se například platforma `Win32` běžně objevuje u 64 bitových systémů Windows.

Nestandardní atributy s informací o platformě

Kromě standardního atributu `platform` vznikly v historii ještě další dva atributy v jiných rodinách prohlížečů, které mají obsahovat podobnou informaci o platformě, na které běží prohlížeč.

- **Gecko** – Implementuje zastaralý atribut `navigator.oscpu`[2].
- **Trident** – Implementuje zastaralý atribut `navigator.cpuClass`[3].

3.1.3 Verze prohlížeče

Atribut `appVersion` [1] musí obsahovat řetězec `"4.0"`, nebo řetězec reprezentující verzi prohlížeče. Matt Frisbie [16] ve své knize uvádí, že tato hodnota řetězce v praxi často nekoresponduje s verzí prohlížeče. Hodnotu atributu jsem proto otestoval napříč rodinami prohlížečů. Následující seznam přiřazuje hodnoty řetězce `appVersion` k příslušným rodinám prohlížečů.

[2] https://developer.mozilla.org/en-US/docs/Web/API/Navigator/oscpu
[3] http://help.dottoro.com/ljjsison.php

- **Blink**, **WebKit**, **Trident**, **EdgeHTML** – Formát odpovídá řetězci `userAgent`, jen bez prefixu `"Mozilla/"`.
- **Gecko** – Formát ve tvaru `MozillaVersion (Platform)`. Hodnoty ve formátu odpovídají hodnotám z řetězce `userAgent`. Mezi příklady hodnot patří: `"5.0 (Windows)"`, `"5.0 (X11)"` a `"5.0 (Macintosh)"`.

3.1.4 Preferované jazyky

Atribut `language` [1] je řetězec reprezentující buď přijatelný[4] jazyk, nebo nejvíce preferovaný jazyk uživatelem. Pole řetězců `languages` reprezentuje všechny preferované jazyky seřazené sestupně dle jejich priority, tj. na prvním místě by se měl vyskytovat jazyk z atributu `language`. Hodnota atributu `languages` by dále měla korespondovat s hodnotami uvedenými v hlavičce `Accept-Language` z protokolu HTTP.

Formát řetězce reprezentující jazyk je definován v rámci specifikace BCP 47[5], kde je nejvíce aktuální specifikací RFC 5646 [46, 22]. Typicky používaný formát pro webové stránky se skládá z povinného kódového označení jazyka a následně dvou volitelných označení. Mezi volitelné označení patří kódové označení skriptu[6] a regionu[7]. Pro jednotlivé části řetězce platí následující:

- **jazyk** musí být 2–3 písmenné slovo typicky psané malými písmeny,
- **skript** musí být 4 písmenné slovo typicky začínající velkým písmenem,
- **region** musí být buď 2 písmenné slovo složené výhradně z alfabetických znaků typicky psaných velkými písmeny, nebo 3 písmenné slovo složené výhradně z číslic.

Například kódové označení jazyka `fr-CA` lze interpretovat jako francouzština používaná v Kanadě.

Preferované jazyky v rodině prohlížečů Trident

Internet Explorer neimplementuje atribut `languages` vůbec a atribut `language` pouze ve verzi 11. Kromě těchto dvou standardních atributů přidává navíc další 3 atributy definující různé jazykové preference.

- **`userLanguage`** – Jazyk reflektuje první nastavený jazyk v konfiguraci prohlížeče a může být kdykoliv změněn uživatelem. Atribut by měl odpovídat atributu `language`.
- **`browserLanguage`** – Jazyk používaný v uživatelském rozhraní prohlížeče. Jazyk je definován již v instalačním souboru prohlížeče a jako takový nemůže být změněn.
- **`systemLanguage`** – Jazyk reflektuje jazyk používaný v operačním systému.

3.1.5 Logické procesory

Atribut `hardwareConcurrency` [16] udává počet logických procesorů dostupných prohlížeči. Hodnota 1 reprezentuje buď minimální počet logických procesorů, nebo indikuje, že počet

[4] Přijatelný jazyk omezuje možnosti identifikace uživatele. Specifikace navrhuje použití jazyku `"en-US"`.
[5] BCP (Best Current Practice) – Zastřešuje specifikace RFC, jejichž verze se mění při jejich aktualizaci.
[6] Skript odlišuje jazyk na základě jeho psané formy nebo dialektu.
[7] Region odlišuje jazyk na základě země, území nebo regionu.

nemohl být zjištěn. Důležité je, že tato hodnota pouze říká, kolik úloh může běžet na pozadí současně v rámci prohlížeče, nikoliv přesný počet logických procesorů, kterými procesor disponuje. Počet logických procesorů tedy může být v zařízení i vyšší, než je uveden.

Počet logických procesorů může být odhadnut pomocí útoku postranním kanálem[8], při kterém se využívá časové analýzy. V principu útok změří čas provádění jedné úlohy na pozadí (angl. Web Worker) a tento čas následně porovnává s časy postupně se zvětšujícím se počtem vytvářených úloh. V momentě, kdy se naměřený čas začne příliš zvyšovat, byl nalezen maximální počet úloh, které mohou běžet na pozadí současně bez snížení výkonu. [9]

3.1.6 Pluginy

Atribut `plugins` [1] reprezentován objektem `PluginArray` obsahuje nainstalované pluginy[9] v prohlížeči. Pluginy v prohlížečích slouží k tomu, aby webová stránka věděla, zda uživatelův prohlížeč podporuje určitý typ obsahu na webové stránce, např. přehrávání videa. V tomto objektu se mohou vyskytovat žádné, některé, nebo všechny nainstalované pluginy. Některé nainstalované pluginy mohou být značeny jako skryté a takové nelze získat enumerací na objektu `plugins`. Dostupnost skrytých pluginů lze však zjistit na základě znalosti jejich jména prostřednictvím funkce `plugins.namedItem`.

Prohlížeč Chrome podporu NPAPI pluginů ukončil v roce 2015. Firefox od verze 52 vydané v roce 2017 podporoval pouze pluginy Adobe Flash, kterým skončila podpora v roce 2020. Prohlížeč Safari nikdy instalaci pluginů nepodporoval. [17, 30]

V dnešní době jsou tak v prohlížečích dostupné výhradně pluginy instalované samotnými prohlížeči, protože ruční instalace pluginů již není oficiálně v prohlížečích podporována.

Detekce operačního systému

Vastel et al. [54] v roce 2018 zjistil, že kromě získání seznamu nainstalovaných pluginů lze dále korelovat jejich přípony souborů s typem operačního systému. Pluginy na systému Windows obsahují příponu `.dll`, na macOS jsou to přípony `.plugin` nebo `.bundle` a na Linuxu zase přípona `.so`.

Dle mého průzkumu již nejsou přípony souboru součástí hodnot atributů na rozhraní objektu `Plugin`. Jediné hodnoty přípon souborů jsou dostupné v atributu `suffixes` na rozhraní objektu `MimeType`, avšak tyto přípony značí pouze typy souborů, se kterými pracuje plugin. Objekty `MimeType` lze získat přímo z objektu `Plugin`, avšak jednodušší způsob, jak získat všechny najednou, je pomocí atributu `navigator.mimeTypes`.

3.1.7 Baterie

Atribut `getBattery` [16] je asynchronní funkce vracející objekt `Promise`[10], který při úspěšném dokončení zpřístupní objekt `BatteryManager`. Objekt `BatteryManager` poskytuje informace o stavu baterie na zařízení uživatele prostřednictvím atributů:

- **`charging`** – Hodnota `false` značí vybíjení baterie. Hodnota `true` značí nabíjení baterie nebo jakoukoliv jinou situaci, např. absenci baterie.

[8] https://github.com/oftn-oswg/core-estimator

[9] Plugin (zásuvný modul) – Aplikace rozšiřující funkcionalitu jiné aplikace, do které lze instalovat i dodatečně.

[10] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

- **`chargingTime`** – Hodnota `0` značí plně nabitou baterii nebo absenci baterie. Hodnota `Infinity` značí vybíjející se baterii. Číselná hodnota značí odhad v sekundách do plného nabití baterie.
- **`dischargingTime`** – Hodnota `Infinity` odpovídá nabíjející se baterii nebo absenci baterie. Číselná hodnota značí odhad v sekundách do plného vybití baterie.
- **`level`** – Hodnota `0` značí vybitou baterii. Hodnota `1` značí plně nabitou baterii, nebo absenci baterie.

Informace o přítomnosti baterie kategorizuje zařízení mezi přenosné a nepřenosné. Časy nabíjení a vybíjení jsou závislé nejen na kapacitě baterie, ale také na aktuálním zatížení zařízení [38]. Informace o stavu baterie mohou být použity pro sledování uživatelů napříč stránkami v krátkém časovém horizontu nebo pro rozlišení uživatelů za prvkem realizující NAT [41]. V době psaní této práce je již atribut označen jako zastaralý a je podporován pouze v rodině prohlížečů Blink[11].

3.1.8 Síťové připojení

Atribut `connection` [16] vrací objekt `NetworkInformation` a poskytuje informace o stavu uživatelova síťového připojení. Atribut je zatím značen jako experimentální a je podporovaný[12] v rodině prohlížečů Blink a ve většině hlavních prohlížečů na mobilních zařízeních. Objekt `NetworkInformation` obsahuje následující atributy:

- **`downlink`** – Odhad šířky pásma v megabitech za sekundu (Mbps). Hodnota je zaokrouhlena na nejbližší násobek 25 kilobitů za sekundu.
- **`effectiveType`** – Řetězec indikující kvalitu a rychlost připojení. Možné hodnoty jsou `"slow-2g"`, `"2g"`, `"3g"` a `"4g"`.
- **`rtt`** – Celé číslo indikující obousměrné zpoždění[13] v milisekundách zaokrouhlené na nejbližších 25 milisekund.
- **`type`** – Řetězec obsahuje informaci o typu spojení, které zařízení využívá ke komunikaci. Mezi nejběžnější typy spojení patří `"bluetooth"`, `"celullar"`, `"ethernet"` nebo `"wifi"`. Hodnota `"none"` znamená žádné připojení, a tudíž by měla být konzistentní se stavem `false` pro atribut `onLine` na objektu `navigator`.
- **`saveData`** – Pravdivostní hodnota indikující, zda uživatel na zařízení nastavil režim omezující využití dat. Na systému iOS se režim nazývá „Režim malého objemu dat“. Informace o tomto režimu je dostupná i v kontextu jazyka CSS pomocí vlastnosti média `"prefers-reduced-data"`, se kterou by měla být konzistentní [24].

3.1.9 Rozložení kláves

Atribut `keyboard` [27] obsahuje objekt `Keyboard` a je značen jako experimentální. Na tomto objektu lze získat asynchronní funkcí `getLayoutMap` rozložení kláves. Rozložení kláves je

[11] https://caniuse.com/mdn-api_navigator_getbattery
[12] https://caniuse.com/mdn-api_networkinformation
[13] Obousměrné zpozdění (Round-Trip Time, RTT) – Doba, za kterou dojde k odeslání zprávy a přijetí potvrzení o jejím doručení.

reprezentováno mapováním fyzických kláves na klávesy dle aktuálně nastaveného rozložení v operačním systému. Objekt `Keyboard` bohužel neposkytuje událost, na základě které by šlo detekovat změnu rozložení kláves v reálném čase vyvolanou uživatelem.

Tento způsob detekce lze nazvat jako pasivní, protože uživatel nemusí nijak interagovat s prohlížečem. Rozložení kláves lze detekovat i aktivním způsobem. Tento způsob spočívá v odposlechu události `keydown` nebo `keyup`. Každá vygenerovaná událost uživatelem obsahuje objekt `KeyboardEvent`. Objekt obsahuje, kromě jiných, také atribut `code` reprezentující fyzicky stisknutou klávesu a atribut `key` reprezentující hodnotu klávesy definovanou nastaveným rozložením kláves v operačním systému.

Pasivní detekce je aktuálně podporovaná[14] pouze v rodině prohlížečů Blink. Prohlížeč Tor Browser vůbec nepodporuje atribut `keyboard` a maže atribut `code` [45]. Prohlížeč Brave má již od roku 2019 nahlášený problém[15] týkající se pasivní detekce, který dodnes nebyl vyřešen.

3.1.10 Atributy detekující hardware a externě připojené zařízení

Následující atributy na objektu `navigator` zpřístupňují informace o hardwaru zařízení, nebo k němu externě připojených zařízení.

- `bluetooth` – Experimentální rozhraní, které je dostupné pouze na zabezpečené komunikaci[16]. Rozhraní implementuje asynchronní funkci `getAvailability`. Pokud uživatel explicitně nezakázal rozhraní, zjistí daná funkce, zda zařízení disponuje hardwarem pro podporu technologie bluetooth.
- `getGamepads` – Experimentální funkce, která vrátí seznam připojených herních ovladačů k zařízení.
- `mediaDevices` – Atribut umožňuje práci s připojenými mediálními zařízeními jako monitory, kamery, mikrofony, sluchátka a podobně. Seznam připojených zařízení lze získat pomocí asynchronní funkce `enumerateDevices`. Prohlížeče Chrome, Edge a Firefox limitují množství poskytovaných informací. Z těchto zkreslených informací lze však odvodit počet dostupných typů zařízení. Prohlížeče Brave a Seznam poskytují nezkreslený list informací. Prohlížeč Tor Browser prozatím odstranil atribut `mediaDevices` úplně [45].
- `xr`[17] – Experimentální rozhraní, které má přinést podporu virtuální reality do prohlížečů. Asynchronní funkce `isSessionSupported` slouží ke zjištění, zda uživatel vlastní zařízení, které dokáže virtuální realitu zobrazit, např. brýle pro virtuální realitu.

3.1.11 Atributy s minimální variabilitou hodnot

Některé atributy na objektu `navigator` mají přesně definované, nebo jinak omezené množství hodnot, kterých mohou nabývat. Z hlediska získávání informací neposkytují samy o sobě příliš diskriminativní informace o uživateli. Jejich význam se však zvyšuje v kombinaci s dalšími informacemi, kde mohou být využity pro detekce nekonzistencí či anomálií v prohlížeči popsaných kapitole 5. Mezi takové atributy se řadí [1, 16]:

[14] https://caniuse.com/mdn-api_keyboard_getlayoutmap

[15] https://github.com/brave/brave-browser/issues/3964

[16] Nezabezpečená komunikace používá ke komunikaci s webovou stránkou pouze protokol HTTP. Zabezpečená komunikace přidává protokol SSL nebo TLS.

[17] https://www.w3.org/TR/webxr

- `appCodeName` – Musí obsahovat řetězec `"Mozilla"`.
- `appName` – Musí obsahovat řetězec `"Netscape"`.
- `cookieEnabled` – Pravdivostní hodnota indikující zda uživatel povolil ukládání souborů cookies. Hodnota `false` odpovídá režimu, ve kterém je zakázané ukládání veškerých souborů cookies. Hodnota `true` značí povolení všech souborů cookies, nebo zakázání souborů cookies třetích stran.
- `deviceMemory` – Experimentální informace reprezentující přibližnou velikost operační paměti v gibibytech. Možné hodnoty v GiB: 0.25, 0.5, 1.2, 4 a 8. Tato informace je dostupná pouze na zabezpečené komunikaci.
- `maxTouchPoints` – Informace udává maximální možný počet simultánních dotykových kontaktů s displejem, které zařízení podporuje. Hodnota větší než 0 indikuje, že zařízení podporuje dotyk [32]. Prohlížeč Tor Browser tento atribut nepodporuje.
- `doNotTrack`[18] – Hodnota reflektuje nastavení prohlížeče „Do Not Track“ (Nesledovat). Cílem tohoto nastavení je zvýšit soukromí uživatele a snížit množství personalizovaného obsahu. Hodnota má dle specifikace tvar `("0" | "1") *DNT-extension`, kde hodnota `DNT-extension` je často vynechávána. V praxi prohlížeče konzistentně uvádí hodnotu `"1"`, pokud uživatel explicitně povolí možnost Nesledovat. Avšak ve výchozím stavu nastavení prohlížeče uvádí rozdílné hodnoty. Prohlížeč Firefox uvádí ve standardním režimu hodnotu `"unspecified"` a v anonymním režimu hodnotu `"1"` [52]. Rodina prohlížečů Blink obsahuje hodnotu `null` v obou režimech. Prohlížeč Internet Explorer ve verzi 9 a 10 implementuje atribut s prefixem `ms`, kde ve verzi 10 je výchozí hodnota nastavena na `"1"` [29]. Internet Explorer 11 neobsahuje atribut `doNotTrack` na objektu `navigator`, nýbrž na objektu `window`.
- `javaEnabled` – Funkce, která indikuje, zda je povolená Java v prohlížeči.
- `onLine` – Pravdivostní hodnota indikující, zda je prohlížeč připojený do sítě. Některé prohlížeče mohou indikovat stav online i přesto, že nemají přístup do internetu, ale pouze v rámci lokální sítě.
- `product` – Musí obsahovat řetězec `"Gecko"`.
- `productSub` – Obsahuje řetězec `"20030107"` pro rodinu prohlížečů Blink, WebKit a EdgeHTML, pro Gecko `"20100101"` a pro Trident `undefined`.
- `vendor` – Odpovídá značce prohlížeče. Pro Blink řetězec `"Google Inc."`, pro WebKit `"Apple Computer, Inc."` a pro Gecko, EdgeHTML a Trident `""`.
- `vendorSub` – Musí obsahovat prázdný řetězec. Hodnota v minulosti měla představovat další informaci o distributorovi prohlížeče.

3.2 Úložiště prohlížeče

Prohlížeče implementují různé typy úložišť, které se liší v reprezentaci uložených dat i době, po kterou jsou data uchovávána. Jedním z hlavních důvodů vzniku těchto různorodých

[18] https://www.w3.org/TR/tracking-dnt

úložišť bylo zvýšení uživatelské přívětivosti webových stránek. Avšak existují i projekty jako `Evercookie`[19], které zneužívají úložiště k perzistentnímu ukládání dat, která jsou typicky použita k identifikaci uživatele. V následujícím seznamu jsou uvedeny různé typy úložišť.

- **Cookies** (`document.cookie`) – Úložiště souborů cookies, které je ve formátu řetězce. Možnosti ukládání souborů cookies musí být konzistentní s nastavenou hodnotou atributu `navigator.cookieEnabled`. Soubory cookies jsou dostupné ze strany serveru i webové stránky.
- **Local** (`window.localStorage`) – Úložiště typu klíč-hodnota. Hodnota je vždy řetězec libovolného tvaru. Uložená data nemají expirační dobu.
- **Session** (`window.sessionStorage`) – Úložiště má stejné rozhraní jako `localStorage`, ale liší se v době, po kterou jsou data ukládána. Data jsou dostupná pouze v jedné záložce a po jejím zavření se smažou.
- **Indexed** (`window.indexedDB`) – Transakční databázový systém podobný relačnímu databázovému systému. Tento typ úložiště je vhodný pro ukládání velkého množství dat, kde data mohou mít podobu i jiných, než pouze primitivních datových typů.
- **Web SQL** (`window.openDatabase`) – Tento typ úložiště měl nabízet podporu SQL dotazů v prohlížeči, avšak rozvoj specifikace[20] se zastavil již v roce 2010. Úložiště je aktuálně podporováno[21] pouze v rodině prohlížečů Blink a v některých mobilních prohlížečích.
- **Cache** (`window.caches`) – Úložiště typu požadavek-odpověď. Při přidávání záznamu se provede požadavek pro specifikovaný záznam a odpověď požadavku je uložena do úložiště. Úložiště funguje pouze na zabezpečené komunikaci a nikdy nebylo podporováno[22] v rodině prohlížečů Trident.

3.2.1 Detekce podpory úložišť

Informace o podpoře úložišť jsou často sbírány, aniž by byl blíže specifikován postup jejich sběru nebo interpretace získaných dat [5, 18]. Webové úložiště definované specifikací Web storage [2] se sestávají z úložišť `sessionStorage` a `localStorage`. Specifikace uvádí, že pro zvýšení soukromí uživatelů mohou být prohlížeči implementovány mechanismy omezující přístup k těmto úložištím. Níže popsané mechanismy mohou ovlivnit způsob i interpretaci získaných informací.

- **Zablokování přístupu k úložištím třetím stranám** – Zamezení přístupu k úložištím webovým stránkám třetích stran, které běží uvnitř prvku `iframe`.
- **Omezení platnosti uložených dat** – Spočívá v automatickém mazání uložených dat, například po zavření prohlížeče.
- **Zacházení s webovými úložišti jako se soubory cookies** – Pokud se uživatel pokusí smazat soubory cookies, smažou se zároveň i data z webových úložišť.

[19] https://github.com/samyk/evercookie
[20] https://www.w3.org/TR/webdatabase
[21] https://caniuse.com/sql-storage
[22] https://caniuse.com/mdn-api_cachestorage

Z mechanismů vyplývá, že sledovací skript třetí strany nacházející se v prvku `iframe` nemusí správně vyhodnotit podporu úložiště a že na dlouhodobé uložení dat v úložištích pro znovu identifikaci se nelze spolehnout. Kromě výše zmíněného má uživatel v prohlížeči možnost explicitně zakázat přístup k úložišti.

Na základě analýzy zabývající se chováním nastavení zákazů úložišť je v tabulce 3.1 zachyceno, zda je možné zakázat nezávisle na sobě úložiště souborů cookies a webové úložiště. Z výsledku je patrné, že explicitní zakázání těchto úložišť lze detekovat právě na základě znalosti jejich chování. Získaná informace odpovídá konfiguraci uživatelova prohlížeče.

	Lze nezávisle zakázat cookies a webové úložiště	**Přístup k úložišti po jeho zákazu**	
		Cookies	**Webové úložiště**
Chrome	Ne	`""`	`DOMException`
Firefox	Ano i Ne		`null`, `DOMException`
Safari	Ne		`DOMException`

Tabulka 3.1: Při zákazů souborů cookies v rozhraní prohlížeče jsou automaticky zakázána i webová úložiště. V prohlížeči Firefox může uživatel zakázat pouze webové úložiště na adrese `about:config` pomocí možnosti `dom.storage.enabled`, kdy po zákazu budou mít přiřazenou hodnotu `null`. Při detekci podpory souborů cookies je nezbytné zkusit vytvořit novou cookie a otestovat, zda se ji podařilo uložit, protože prázdný řetězec může reprezentovat i stav, kdy nejsou momentálně žádné cookies uloženy. Detekce podpory webových úložišť musí být ošetřena, protože při jejich zákazu v rozhraní prohlížeče bude při přístupu vyvolána výjimka.

3.3 Zobrazovací zařízení

Informace o vlastnostech zobrazovacích zařízení, spolu s rozměry okna prohlížeče, napomáhají webovým stránkám vybrat vhodný formát obsahu ušitý na míru každému uživateli. Pod vhodný obsah lze zahrnout například média, jako obrázky nebo videa, u kterých se může měnit jejich velikost, rozlišení či formát.

Na rozhraní objektu `window` se vyskytují 4 standardizované atributy, které nabízí informace o zobrazovacím zařízení. Tyto atributy jsou [47, 16]:

- `devicePixelRatio` – Značí poměr mezi velikostí jednoho pixelu CSS[23] na aktuální úrovni přiblížení stránky a vertikální velikosti jednoho fyzického pixelu. Například pokud mobilní zařízení disponuje fyzickým rozlišením 750×1334 a CSS rozlišením pouze 375×667, pak hodnota atributu `devicePixelRatio` se rovná 2, což značí, že 1 pixel CSS bude vykreslen na mřížce 2×2 a zabere celkem 4 fyzické pixely [35]. Na zařízeních, kde uživatel může změnit úroveň přiblížení stránky (angl. page zoom), se velikost jednoho pixelu CSS přepočítává na základě aktuální úrovně přiblížení. Pokud se tedy při úrovni přiblížení 100 % hodnota atributu `devicePixelRatio` rovná 1, pak při přiblížení 200 % se bude rovnat 2. Výjimka nastává v prohlížeči Safari, kde se hodnota nemění v závislosti na úrovni přiblížení stránky, ale je vždy konstantní.

[23] Pixel CSS (též značen jako logický pixel) – značí míru pixelu, který je měřen v jednotkách `px`. Hodnota `1 px` odpovídá přibližně `1/96 palce`.

- **`screenX`** (alias **`screenLeft`**) – Atribut vrací relativní horizontální vzdálenost v CSS pixelech mezi aktuální pozicí levého okraje prohlížeče a levým okrajem zobrazovacího zařízení. Specifikace však již neuvádí, že na systému Windows se vzdálenost počítá vždy od zobrazovacího zařízení, které je nastavené jako primární. Na systému Ubuntu zase na primárním nastavením nezáleží a vše se odvíjí pouze od nastaveného rozložení monitorů.
- **`screenY`** (alias **`screenTop`**) – Analogicky k atributu **`screenX`**, pouze v kontextu relativní vertikální vzdálenosti mezi horními okraji prohlížeče a zobrazovacího zařízení.
- **`screen`** – Atribut implementuje objekt s rozhraním **`Screen`** a je popsán dále.

Atribut **`screen`** obsahuje informace o obrazovce, na které se vyskytuje okno prohlížeče. Následující atributy jsou definovány v rámci specifikace.

- **`width`** – Šířka zobrazovacího zařízení v CSS pixelech.
- **`height`** – Výška zobrazovacího zařízení v CSS pixelech.
- **`availWidth`** – Šířka prostoru, kterou má prohlížeč k dispozici. Podobně jako atribut **`width`** udává šířku zobrazovacího zařízení v CSS pixelech, ale bez prvků operačního systému, např. bez šířky hlavního panelu ve Windows za předpokladu, že je panel situován vertikálně. Pokud je prohlížeč maximalizován, pak platí rovnost **`screen.availWidth = window.outerWidth`**. Na základě porovnání hodnot atributů **`width`** a **`availWidth`** lze odvodit, jakým způsobem má uživatel nastavené prostředí v operačním systému, resp. přítomnost či pozici hlavního panelu.
- **`availHeight`** – Analogicky k atributu **`availWidth`**, pouze v kontextu dostupné výšky prohlížeče.
- **`colorDepth`** – Číslo reprezentující barevnou hloubku. Barevná hloubka se udává v počtu bitů na pixel, kde bity popisují barvu pixelu. Specifikace udává, že tato hodnota musí být 24, která je také označována jako True color.
- **`pixelDepth`** – Tento atribut reprezentuje stejnou informaci jako **`colorDepth`** a měl by vracet pouze hodnotu 24.
- **`orientation`**[24] – Atribut implementuje rozhraní **`ScreenOrientation`**. Nejzajímavější atribut z hlediska zisku informací a dostupné podpory je **`type`**.
  - **`type`** – Atribut reprezentuje typ orientace zobrazovacího zařízení. Hodnota tohoto atributu může být použita pro ověření správnosti šířky a výšky zařízení, protože například hodnota **`portrait`** může nastat pouze pokud platí šířka≤výška.

V následujícím seznamu jsou atributy, které nejsou definovány v rámci specifikace. Tyto atributy jsou však široce podporovány[25] napříč prohlížeči a poskytují další zajímavé informace.

[24] https://www.w3.org/TR/screen-orientation

[25] https://caniuse.com/mdn-api_screen_availleft, caniuse.com/mdn-api_screen_availtop

- `availLeft` – Hodnota tohoto atributu je rovna hodnotě `screenX` při maximalizovaném okně prohlížeče. Jedná se tedy o absolutní vzdálenost, při které nezáleží na relativní pozici okna prohlížeče na zobrazovacím zařízení. Informace z tohoto atributu je hodnotnější, protože není zatížena pohybem okna prohlížeče jako atribut `screenX`.
- `availTop` – Analogicky k atributu `availLeft`, pouze v kontextu absolutní vertikální vzdálenosti.

3.3.1 Detekce obnovovací frekvence obrazovky

Funkce `requestAnimationFrame` [13] dostupná na rozhraní `window` slouží k naplánování úloh, které se v budoucnu provedou. Všechny aktuálně naplánované úlohy se provádí v části smyčky událostí, která je zodpovědná za vykreslování webové stránky. Správné načasování doby provádění těchto úloh je nezbytné pro plynulý chod webových stránek. Snaha prohlížeče je taková, aby frekvence provádění úloh odpovídala obnovovací frekvenci zobrazovacího zařízení.

Pokud zobrazovací zařízení disponuje obnovovací frekvenci 60 Hz, pak je schopno překreslit obraz až 60× za jednu sekundu. Prohlížeč se tedy bude snažit vykonávat naplánované úlohy zhruba každých $16.\overline{6}$ ms. Pokud prohlížeč nezvládá tuto frekvenci dodržet, pak může snížit frekvenci vykreslování na nižší udržitelnou hodnotu, nebo přeskočit určité vykreslování. Mezi důvody pro snížení frekvence patří dlouhá fronta mikroúloh[26], vnořený kontext procházení[27], zejména pocházející od třetí strany, nebo jiné dlouho běžící úlohy.

3.3.2 Detekce kvality zobrazovacího zařízení

Ve 4. a 5. úrovni specifikace Media Queries [49, 24] je specifikována nová vlastnost média `"color-gamut"`, která definuje přibližný rozsah barev podporovaný na zobrazovacím zařízení. Tato vlastnost může nabývat jedné z uvedených hodnot seřazených od nejhorší k nejlepší `srgb`, `p3`, nebo `rec2020`.

V rámci nejnovějšího návrhu specifikace CSSOM View Module [47] z roku 2020 se objevil následující výpisek 3.1, jehož cílem je detekovat HDR obrazovku.

```
screen.colorDepth >= 48 && window.matchMedia("(color-gamut: p3)").matches;
```

Výpis 3.1: Detekce HDR obrazovky.

Dle specifikace se očekává, že většina obrazovek podporuje alespoň `srgb` gamut. Starší typy zařízení, jako monochromatické monitory, však nemusí podporovat ani tento a pro jejich detekci lze použít dotaz `"(not (color-gamut)"`.

3.3.3 Detekce podpory dotyku na zobrazovacím zařízení

Podpora dotyku lze detekovat prostřednictvím jazyka JavsScript a CSS. Detekce podpory v jazyce JavaScript již byla publikována [54], avšak detekce v jazyce CSS, kterou zde představuji, zatím, dle mých znalostí, publikována nebyla.

[26] Mikroúloha (angl. microtask) je zpětné volání z rozhraní `Promise` nebo `MutationObserver`.

[27] Vnořený kontext procházení (angl. nested browsing contexts) je kontext vytvořený například prvkem `iframe`.

Detekce podpory dotyku v jazyce JavaScript

1. Detekce maximálního počtu simultánních dotykových kontaktů prostřednictvím atributu `navigator.maxTouchPoints` z kapitoly 3.1.11. Z hlediska kompatibility lze k tomuto atributu přidat i prefix `ms`[28], který zajistí podporu pro Internet Explorer 10.

2. Detekce přítomnosti atributu `ontouchstart` na rozhraní `window`. K tomuto atributu existuje návrh specifikace [51], který detekci demonstruje.

3. Detekce možnosti vytvoření umělé události pro rozhraní `TouchEvent` prostřednictvím funkce `document.createEvent`[29]. Rozhraní by mělo být dostupné pro zařízení, které podporuje dotyk [51]. Pokud rozhraní není dostupné, je vyvolána výjimka.

Detekce z bodu 2. a 3. spoléhají na návrh specifikace [51], který není akceptován jako standard konsorcia W3C. Princip fungování těchto detekcí se může v budoucnu změnit.

Detekce podpory dotyku v jazyce CSS

1. Vlastnost média `hover`, která nabývá jedné z hodnot `none` a `hover`. Hodnota je určena prohlížečem na základě toho, zda má uživatel možnost provádět operaci `hover`.

2. Vlastnost média `pointer`, která nabývá jedné z hodnot `none`, `coarse` a `fine`. Hodnota je určena na základě přítomnosti a přesnosti zařízení, které je použito k interakci s webovou stránkou.

Ve specifikacích [49, 24] je uvedena tabulka 3.2, která na základě kombinací hodnot uvedených vlastností médií detekuje typ zařízení, kterým uživatel interaguje s webovou stránkou.

	`pointer: none`	`pointer: coarse`	`pointer: fine`
`hover: none`	klávesnice, navigace pomocí operace focus	chytrý telefon, dotyková obrazovka	běžný stylus (Cintiq, Wacom, ...)
`hover: hover`	—	Nintendo Wii, Kinect	myš, touchpad, pokročilý stylus (Surface, Samsung Note, Wacom Intuos Pro, ...)

Tabulka 3.2: Detekce typu zařízení na základě vlastností médií `hover` a `pointer`.

Z tabulky 3.2 vyplývá, že zobrazovací zařízení podporuje dotyk, pokud současně splňuje vlastnosti média `hover: none` a `pointer: coarse`.

3.4 Internacionalizace a časové informace

Internacionalizace v kontextu webových stránek znamená jejich schopnost přizpůsobit se jazykovým a kulturním konvencím používanými různými lidskými jazyky či zeměmi [6].

[28] https://caniuse.com/mdn-api_navigator_maxtouchpoints
[29] https://dom.spec.whatwg.org/#dom-document-createevent

V praxi webové stránky mohou používat například formátování čísel nebo času prostřednictvím rozhraní `Intl`[30]. Informace o času jsou dostupné na rozhraní `Date`[31] [19], na kterém lze navíc realizovat internacionalizace podobně jako na rozhraní `Intl`.

3.4.1 Časové pásmo

Rozhraní `Date` implementuje funkci `getTimezoneOffset`, která spočítá rozdíl v minutách mezi koordinovaným světovým čas (angl. Coordinated Universal Time, UTC) a aktuálně nastaveným časovým pásmem v uživatelově operačním systému. Pokud má například uživatel nastavené časové pásmo `UTC+01:00`, tzn. čas je posunut o 60 minut dopředu oproti UTC, pak je výsledek funkce roven `-60`.

Rozhraní `Intl` implementuje konstruktor `DateTimeFormat`, na kterém existuje funkce `resolvedOptions`. Výsledkem volání této funkce je objekt, který obsahuje také atribut `timeZone`. Hodnota atributu je typu řetězec ve formátu `"oblast/lokace"`, který koresponduje s formátem pásma v databázi časových pásem IANA[32]. Oblast je typicky jméno kontinentu nebo oceánu a lokace definuje jméno města nebo ostrova v rámci dané oblasti. Například časové pásmo `UTC+01:00` může například nabývat jedné z hodnot `"Europe/Prague"`, `"Europe/Warsaw"`, nebo `"Europe/Paris"`. Případné mezery v názvu lokace jsou nahrazeny znakem `"_"` a tečky jsou z názvu vynechány úplně.

3.4.2 Detekce letního času

Výpočet funkce `getTimezoneOffset` závisí na nastaveném časovém pásmu v operačním systému, ale i na posunu daném letním časem (angl. Daylight Saving Time, DST) [19].

- V České republice platí letní čas od poslední neděle v březnu do poslední neděle v říjnu.
- V USA a Kanadě platí letní čas od druhé neděle v březnu do první neděle v listopadu.

Princip detekce letního času spočívá ve výběru dvou dat, kde jedno datum bude spadat do letního času a druhé nebude. Pokud rozdíl posunů zvolených dat je nenulový, pak je detekováno použití letního času. Následující výpisek 3.2 demonstruje detekci letního času.

```
const UTCOffset = new Date(2020, 0, 1).getTimezoneOffset(); // 1.1.2020
const DSTOffset = new Date(2020, 6, 1).getTimezoneOffset(); // 1.7.2020
// If DST is used, then difference will be equal to DST and not to zero.
DSTOffset - UTCOffset !== 0 ? "Does use DST" : "Does not use DST";
```

Výpis 3.2: Obecná detekce letního času.

3.4.3 Jazyk hostitelského prostředí

Jazyk hostitelského prostředí lze získat z atributu `locale`, který je taktéž dostupný po zavolání funkce `new Intl.DateTimeFormat().resolvedOptions()`. Hodnota atributu odpovídá jazyku, do jakého je přeloženo uživatelské rozhraní prohlížeče.

[30] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Intl
[31] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date
[32] https://www.iana.org/time-zones

3.5 Hlavičky protokolu HTTP

Hlavičky protokolu HTTP nabízí další způsob, jak předat informace mezi klientem a serverem v rámci požadavku nebo odpovědi [39, 14]. Získávání informací o uživateli na základě hlaviček se nazývá pasivní způsob získávání informací, protože k získaní těchto informací není třeba jazyk JavaScript [28]. Hlavičky jsou ve formátu `klíč:hodnota`.

Protokol HTTP umožňuje přenášet i hlavičky, které nebyly specifikované v rámci žádného standardu. Historicky tyto hlavičky obsahovaly prefix `X-`, avšak tato praxe byla v roce 2012 uvedena jako zastaralá[33]. Protokol HTTP/1.1 uvádí různé pokyny pro to, jak by správně měla vypadat specifikace nové hlavičky.

V této kapitole budou představeny zejména ty hlavičky protokolu HTTP, které obsahují informace týkající se uživatele, nebo které mohou být zneužity ke sledování uživatele.

3.5.1 Hlavičky pro vyjednávání obsahu

Tento typ hlaviček obsahuje informace o obsahu, kterému prohlížeč rozumí a preferuje. Server by na základě těchto informací měl vybrat z jemu dostupného obsahu tu nejlepší variantu. V praxi to tedy znamená, že pokud server obsahuje různé formáty stejného obrázku, různé překlady webové stránky nebo implementuje více způsobů komprese přenášených dat, tak by měl použít vždy tu nejlepší možnou variantu, resp. kombinaci, dle svých možností.

Mnoho hodnot hlaviček může používat parametr `"q"`, známý také jako `qvalue`. Hodnota parametru definuje prioritu preference daného typu obsahu. Priorita je číslo od 0 do 1, kde hodnota 0 znamená neakceptovatelný obsah a hodnota 1 nejvíce preferovaný obsah. Pokud parametr `"q"` není pro daný obsah definován, pak je jeho hodnota rovna 1.

- **`Accept`** – Hlavička specifikuje, jakým typům obsahu, ve formátu typů MIME[34], prohlížeč rozumí. Příklad zjednodušené hodnoty: `text/html, image/webp, */*;q=0.8`.
- **`Accept-Charset`** – Hlavička specifikuje, jakému kódování znaků prohlížeč rozumí. Většina prohlížečů tuto hlavičku neposílá[35], aby omezila míru prozrazených informací. Příklad hodnoty: `utf-8, iso-8859-1;q=0.5`.
- **`Accept-Encoding`** – Hlavička specifikuje, jakým kompresním algoritmům dat prohlížeč rozumí. Příklad hodnoty: `gzip, deflate, br`.
- **`Accept-Language`** – Hlavička specifikuje, jaké jazyky uživatel preferuje. Hodnota hlavičky by měla odpovídat hodnotě atributu `navigator.languages`, jak bylo uvedeno v kapitole 3.1.4. Příklad hodnoty: `cs, en-GB;q=0.9, en;q=0.8`.

3.5.2 Hlavičky obsahující kontext požadavku

Hlavičky v této kategorii obsahují dodatečné informace o uživateli a jeho kontextu procházení webových stránek.

- **`From`** – Hlavička obsahuje e-mailovou adresu uživatele obsluhujícího prohlížeč. Odeslání této hlavičky by mělo být podmíněno explicitním povolením od uživatele. Prohlížeče ovládané pomocí robotů by měly tuto hlavičku odesílat vždy, aby v případě jimi způsobených komplikací na serveru mohli být autoři robota kontaktováni.

[33] https://tools.ietf.org/html/bcp178
[34] https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types
[35] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Charset

- **`Referer`** – Hlavička obsahuje URI referenci na zdroj, ze kterého byl získán cílový zdroj. Hodnota této hlavičky umožňuje generovat zpětné odkazy, které mohou být použity pro mapování pohybu uživatele napříč webovými stránkami.
- **`User-Agent`** – Hlavička by měla obsahovat stejný řetězec, jako je dostupný na atributu `navigator.userAgent`, který je popsán v kapitole 3.1.1.

3.5.3 Hlavička DNT

Hodnota hlavičky `DNT` (Do Not Track) reflektuje nastavení uživatele týkající se jeho preference, jestli si přeje být sledován. Hodnota hlavičky by měla korespondovat s hodnotou atributu `navigator.doNotTrack`, který byl uveden v kapitole 3.1.11.

3.5.4 Nové hlavičky pro vyjednávání obsahu

Tyto nové hlavičky známé jako `Client Hints` [55, 56] mají sloužit, stejně jako hlavičky s prefixem `Accept` z předchozí kapitoly 3.5.1, k tomu, aby server na základě jimi zaslaných informací dokázal vybrat nejvhodnější možnou formu obsahu, kterou zašle zpět klientovi. Hlavičky `Client Hints` se však od hlaviček `Accept` velmi odlišují a nejdůležitější rozdíly jsou proto zachyceny v tabulce 3.3.

První návrh specifikace hlaviček `Client Hints` vznikl již v roce 2013[36], ale doposud neexistuje žádný standard pro tyto hlavičky. Přesto jsou některé hlavičky podporovány[37] v prohlížeči Chrome již od verze 46 publikované v roce 2015.

	Standardní hlavičky `Accept`	**Nové hlavičky `Client Hints`**
Informace	O prohlížeči.	O zařízení, zobrazovacím zařízení a stavu připojení do sítě.
Dostupnost	Automaticky dostupné na každém požadavku.	Nejprve je o ně nutné požádat. Poté mohou být dostupné na následujících požadavcích.
Dostupné třetí straně	Ano.	Ne, musí být explicitně povoleno.
Nezabezpečená komunikace	Ano.	Ne. Pouze přes HTTPS.
Standard	HTTP/1.0 a HTTP/1.1	Ne.

Tabulka 3.3: Porovnání standardních a nových hlaviček pro vyjednávání obsahu.

Následující seznam obsahuje názvy hlaviček `Client Hints`, které mapuje na zdroje informací získatelných z jazyka JavaScript, jak bylo uvedeno v předchozích kapitolách.

- **Hlavičky odpovídající hodnotě atributu `navigator.userAgent`** - Všechny tyto hlavičky začínají prefixem `Sec-CH-` a jsou to: `UA`, `UA-Arch`, `UA-Model`, `UA-Platform`, `UA-Platform-Version`, `UA-Full-Version` a `UA-Mobile`.
- **Informace o zařízení**

[36] https://datatracker.ietf.org/doc/draft-ietf-httpbis-client-hints/00
[37] https://caniuse.com/client-hints-dpr-width-viewport

  - `Device-Memory` – `navigator.deviceMemory`

- **Informace o zobrazovacím zařízení** – obě hodnoty mohou být ovlivněny aktuální úrovní přiblížení stránky.
  - `DPR` – `window.devicePixelRatio`
  - `Viewport-Width` – `window.innerWidth`

- **Informace o stavu připojení**
  - `Downlink` – `navigator.connection.downlink`
  - `ECT` – `navigator.connection.effectiveType`
  - `RTT` – `navigator.connection.rtt`
  - `Save-Data` – `navigator.connection.saveData`

Význam hlaviček `Client Hints`

Hlavičky mají potenciál významně omezit množství přenášených dat na internetu a ještě více upravit uživatelské rozhraní na míru každému uživateli. Z hlediska získávání informací neposkytují žádné nové informace o uživateli. Nicméně získané informace mohou být využity k detekci nekonzistencí, protože by vždy měly odpovídat hodnotám získaným pomocí jazyka JavaScript.

3.5.5 Získání dalších informací z hlaviček protokolu HTTP

Laperdrix et al. [30] potvrdil, že kromě hlaviček zmíněných v této kapitole se v požadavcích také vyskytují hlavičky ovlivněné nebo nastavené jinými programy nebo webovými rozšířeními. Celkem pozoroval 222 různých hlaviček, které tvořily 1 182 různých seznamů hlaviček. Například bylo pozorováno následující:

- Prohlížeč Opera na chytrých mobilech přidává hlavičku `X-OperaMin-Phone-UA`. Prohlížeč Puffin přidává hlavičku `X-Puffin-UA`.
- Webové rozšíření FirePHP v prohlížeči Firefox přidává hlavičku `x-FirePHP-Version` a `x-FirePHP`.
- Hlavičky také mohou být přidány některými proxy servery.

3.6 Písma

Pro zobrazení textu používají webové prohlížeče písma. Písma lze do webové stránky importovat dynamicky za běhu a nebo lze využít již předem nainstalovaných písem v operačním systému. Existuje seznam šesti písem bezpečných pro web (angl. Web safe fonts)[38], které by měly být dostupné na nejvíce používaných operačních systémech jako Windows, macOS, distribuce Linuxu, Android a iOS. Tato písma pochází z rodin písem `serif`, `sans-serif` a `monospace`.

[38] https://developer.mozilla.org/en-US/docs/Learn/CSS/Styling_text/Fundamentals

3.6.1 Detekce písem pomocí technologií ActiveX a Adobe Flash

Získání všech písem je možné již od 8. verze prohlížeče Internet Explorer[39] poprvé vydané v roce 2009. Získání úplného seznamu písem v rodině prohlížečů Trident je založeno na technologii ActiveX, která pochází od společnosti Microsoft. Princip získání písem je takový, že na vytvořeném objektu ActiveX je dostupný atribut `fonts`. Tento atribut obsahuje počet všech písem (`fonts.Count`) a na základě indexu lze voláním `fonts(index)` přistoupit k názvu písma.

V roce 2010 Eckersley [10] poukázal, že písma jsou jednou z nejvíce unikátních informací, které mohou být použity pro identifikaci uživatele. Eckersley vycházel z téměř půl milionu otisků pocházejících z projektu Panopticlick[40]. Veškerá písma byla získána pomocí technologií Adobe Flash (dále jen „Flash“) nebo Java Virtual Machine (dále jen „Java“). Obě tyto technologie vrací kompletní seznam písem. Eckersley dále potvrdil hypotézu, že pořadí písem v získaném seznamu je neměnné v čase, a tudíž že přidává další dodatečnou informaci.

V roce 2016 Laperdrix et al. [30] našel 221 804 odlišných písem, které tvořily 36 202 unikátních seznamů písem. Bylo zjištěno, že uživatele systémů Windows a macOS mívají v průměru 2–3× více písem než uživatelé na systému Linux. Tyto seznamy písem byly také získány pomocí technologie Flash.

Technologii Flash skončila podpora[41] k 31.12.2020 a technologie ActiveX vždy fungovala[42] nativně pouze v rodině prohlížečů Trident.

3.6.2 Detekce písem pomocí jazyka JavaScript

Již v roce 2013 Nikiforakis et al. [40] představil princip detekce přítomnosti písma pomocí jazyka JavaScript. Princip detekce je založen na faktu, že různá písma mají různé velikosti. Pokud se na prvek HTML, který obsahuje text, aplikuje písmo a prvek změní svoji velikost, pak je to díky dostupnosti písma. Pro detekci je nezbytné definovat:

1. rodinu písma, kterou prohlížeče podporují, např. `sans-serif`,
2. seznam písem, pro něž mát být detekována dostupnost,
3. funkci, která spočítá rozměry písma. Tato funkce musí vytvořit prvek `span`, do kterého vloží text např. `"font_detection"`. Na prvek aplikuje detekované písmo včetně záchranného výchozího písma a nastaví písmu velikost 72 px. Následně funkce vloží prvek do stránky a spočítá jeho velikost. Velikost písma musí být velká, aby bylo možné zachytit i nejmenší rozdíly ve způsobu vykreslení různých písem.
4. funkci, která provede vyhodnocení dostupnosti písma. Písmo je dostupné, pokud se liší velikosti detekovaného písma a výchozího písma `sans-serif`. Pokud se velikosti neliší, pak písmo nebylo nalezeno a pro vykreslení textu bylo použito výchozí písmo.

Důvod použití přístupu pomocí jazyka JavaScript spočívá v nahrazení již zastaralých technologií ActiveX a Flash. Oproti zastaralým technologiím má tento přístup několik nevýhod:

[39] http://help.dottoro.com/ljpgiwbg.php
[40] https://panopticlick.eff.org
[41] https://www.adobe.com/cz/products/flashplayer/end-of-life.html
[42] https://en.wikipedia.org/wiki/ActiveX#ActiveX_in_non-Internet_Explorer_applications

- je výpočetně náročnější, protože mnoho výpočtů musí být provedeno přímo u klienta,
- musí být definován vhodný seznam písem, takže nelze zaručit, že budou vždy detekována všechna písma,
- dochází ke ztrátě informace v podobě pořadí písem,
- detekce může být ovlivněna písmy načtenými webovými aplikacemi.

V roce 2017 Kobusińska et al. [29] porovnala detekci písma založenou na jazycích JavaScript a CSS oproti metodě detekce písem založené na prvku canvas. Metoda využívající prvek canvas byla pomalejší a detekovala méně písem. Mezi další zajímavé poznatky patří zejména:

- text `"adfgjlmrsuvwwwwz7901"` vykazuje větší rozdíly v písmech nežli texty založené zejména na znacích `m` a `w`,
- velikost textu 70 px měla téměř shodné výsledky s velikostmi 180 a 200 px,
- výchozí písmo `monospace` bylo vykresleno v průměru až 10× rychleji než písmo `sans-serif`,
- míra získané informace byla založena na detekci 821 různých písem. S třetinovou ztrátou míry informace bylo možné optimalizovat počet písem na 9[43], čímž se snížila doba výpočtu v řádech jednotek sekund.

V roce 2018 Gómez-Boix et al. [17] zjistil, že mezi jím 66 detekovanými písmy[44] existovala taková, která měla stejné rozměry jako výchozí písmo. Jako příklad bylo uvedené detekované písmo Tinos a výchozí písmo Times New Roman z rodiny písem `serif`. Použitý text byl zvolen jako `"TimesNEWRoman"` a velikost písma nebyla blíže specifikována. Po bližším prozkoumání jsem však zjistil, že uvedená písma, ač přes jejich velkou podobnost, lze rozlišit na základě jejich rozměrů (obrázek 3.1b). Možná chyba jejich měření mohla nastat například zvolením příliš malé velikosti písma (obrázek 3.1a).

Studie se nesnaží optimalizovat text prvku ani velikost písma, ale nabízí jiné řešení. Řešení spočívá v definovaní 3 různých výchozích písem. Tato písma jsou z rodin `serif`, `sans-serif` a `monospace`. Pro každé detekované písmo jsou tedy vytvořeny 3 varianty, kde každá varianta obsahuje jedno záchranné výchozí písmo. Písmo je dostupné, pokud se alespoň jedna varianta detekovaného písma liší rozměry od daného výchozího písma. Tento způsob řešení sice omezil falešně negativní výsledky dostupnosti, ale je nutné vytvořit 3× více prvků, což v nejhorším případě vede až na 3× více porovnání.

3.7 Zvuk

Zpracování zvukového signálu je nedílnou součástí zejména multimediálních webových aplikací. V dnešní době je možné využívat pokročilého zpracování zvukového signálu rozhraními definovanými ve specifikaci Web Audio API [4]. Rozhraní byla navržena tak, aby podporovala řadu případu užití[45]. K podpoře řady případů užití je nutné použít hardwarové komponenty zařízení.

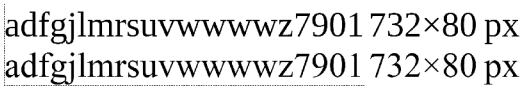
[43] Open Sans, Brush Script MT, Estrangelo Edessa, Gadugi, Roman, Papyrus, MT Extra, Wingdings, Segoe UI Semibold

[44] Seznam detekovaných písem je uveden v příloze A dané studie.

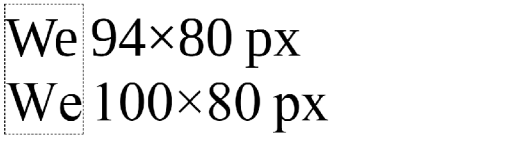
[45] https://www.w3.org/TR/webaudio-usecases

(a) Velikost písma 14 px. V tomto případě se nepodařilo detekovat písmo, protože písma dosahují stejných rozměrů.

(b) Velikost písma 72 px. V tomto případě se podařilo detekovat písmo, protože šířka písem se liší o 3 px.



(c) Velikost písma 72 px. Kobusińska et al. [29] navrhla tento text, protože by měl vykazovat větší rozdíly mezi písmy. V tomto případě bohužel selhal.



(d) Velikost písma 72 px. Mnou navržený text `"We"` vykazuje v tomto případě nejlepší výsledky, protože je nejkratší a lze na něm pozorovat největší rozdíl velikostí. Je však zřejmé, že tento text nebude fungovat vždy.

Obrázek 3.1: Porovnání vlivu velikosti písma a porovnávaného textu na rozměry podobných písem. Horní písmo je Tinos a spodní Times New Roman. Rozměry mají tvar šířka×výška. Na obrázku lze pozorovat, že rozmanitější a delší text nemusí nutně vždy znamenat lepší výsledek. Jedno z možných vysvětlení může být, že drobné rozdíly v písmenech mezi písmy se negují, což zapříčiní stejnou velikost zvoleného textu. Velikost písma je však velmi důležitá, a čím je písmo větší, tím větší rozdíly lze pozorovat. Při velmi malé velikosti písma bývají rozdíly nulové a při příliš velkých velikostech se rozdíly v písmech již nezvětšují.

3.7.1 Získání informací ze zpracování zvukového signálu

Englehardt et al. [11] jako první v roce 2016 zveřejnil techniku získávání informací, jejíž princip je založen na generování zvukového signálu a jeho následného zpracování. Výsledek zpracování signálu by měl být vždy stejný pro stejné zařízení a prohlížeč, ale rozdílný mezi více prohlížeči na stejném zařízení nebo mezi jinými zařízeními. Dle mých pozorování je navíc výsledek stejný i mezi různými prohlížeči ze stejné rodiny prohlížečů na stejném zařízení.

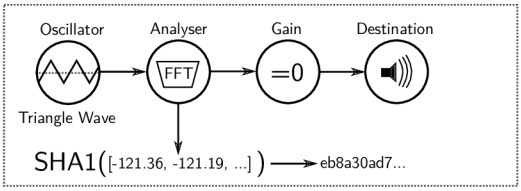
Generování periodického průběhu signálu je umožněno uzlem, který je specifikován rozhraním `OscillatorNode`. Rozhraní umožňuje nastavit sinusový, čtvercový, pilovitý, trojúhelníkový nebo vlastní průběh signálu. Vygenerovaný signál může být dále zpracováván dalšími uzly za účelem zvýšení míry získané informace, která je způsobena jejich specifickým způsobem zpracování. Specifický způsob zpracování je dán hardwarovým, ale i softwarovým vybavením zařízení. Studie uvádí zpracování signálu pomocí rozhraní `AudioContext` (obrázek 3.2a) a `OfflineAudioContext` (obrázek 3.2b). Kód pro oba způsoby zpracování signálu je dostupný na webové stránce[46] autorů studie.

3.7.2 Získání informací o konfiguraci zvukového systému

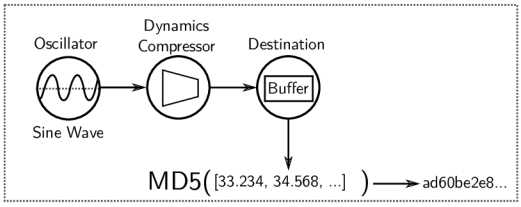
Kromě informací získaných zpracováním zvukového signálu lze z rozhraní `AudioContext` získat atributy, které jsou popsané v následujícím seznamu.

- `baseLatency` – Značí zpoždění v sekundách, které je způsobené předáním zvuku z rozhraní `AudioDestinationNode` do zvukového systému zařízení.

[46] `view-source:https://audiofingerprint.openwpm.com`



(a) Trojúhelníkový průběh signálu, generovaný uzlem `OscillatorNode`, je předán do uzlu `AnalyserNode`, který je schopen provést analýzu signálu ve frekvenční a časové doméně pomocí Rychlé Fourierovy transformace (angl. Fast Fourier transform, FFT). Uzel `AnalyserNode` je dále napojen na uzel `ScriptProcessorNode`, který umožňuje záchyt zpracovávaného signálu z uzlu `AnalyserNode` událostí `onaudioprocess`. Uzel `ScriptProcessorNode` je napojen na uzel `GainNode`, který ztlumí jeho vstup, aby uživatel nic neslyšel. Uzel `GainNode` je nakonec napojen na rozhraní `AudioContext`, odkud bude předán zvuk na výstup, např. reproduktory, a je zahájeno generování signálu uzlem `OscillatorNode`.



(b) Sinusový průběh signálu, generovaný uzlem `OscillatorNode`, je předán do uzlu `DynamicsCompressorNode`, který provede efekt komprese dynamiky signálu, tj. nejslabší části signálu budou zesíleny a nejsilnější části signálu budou zeslabeny. Celkový zvuk signálu by měl být po aplikaci tohoto efektu bohatší a plnější. Uzel `DynamicsCompressorNode` je nakonec napojen na rozhraní `OfflineAudioContext`, kde je po dokončení efektu komprese dynamiky dostupný výsledek jako parametr události `oncomplete`. Výsledek je objekt, na kterém je atribut `renderedBuffer` typu `AudioBuffer` zpřístupňující data signálu po zavolání funkce `getChannelData`. V tomto případě nedochází k přehrání zvuku.

Obrázek 3.2: Možné způsoby generování a zpracování zvukového signálu jehož výsledkem je otisk použitelný pro identifikaci uživatele (převzato z [11]).

- **`outputLatency`** – Odhad zpoždění v sekundách, které reflektuje dobu od vyžádání přehrání zvuku do doby přehrání prvního vzorku.
- **`sampleRate`** – Vzorkovací frekvence signálu.
- **`destination`** – Atribut je specifikován rozhraním `AudioDestinationNode`, které reprezentuje poslední uzel, ze kterého je zvuk předán na výstupní zařízení. Na rozhraní jsou dostupné atributy `channelCount`, `channelCountMode`, `channelInterpretation`, `maxChannelCount`, `numberOfInputs` a `numberOfOutputs`.

3.8 Grafické vlastnosti prvku canvas

Prvek canvas [16, 37] zapsaný kódem `<canvas>` představuje oblast webové stránky, na kterou lze vykreslit rastrová grafika prostřednictvím jazyka JavaScript. Před začátkem vykreslování je nutné získat odpovídající grafický kontext poskytující rozhraní pro manipulaci vykreslovaného obsahu. Mezi důležité grafické kontexty se řadí kontext `"2d"` pro vykreslení 2D obsahu a kontext `"webgl"`, nebo pokud je podporován `"webgl2"`, pro vykreslení 2D a 3D obsahu. Pokud v rámci této práce nebude uvedeno jinak, bude nadále uvažováno získání kontextu `"2d"`, který je definován ve specifikaci HTML5[47].

Velikost oblasti prvku canvas a tedy i velikost vykresleného obsahu má významný vliv na kvalitu získaných informací. Čím větší je oblast, a tím pádem i vykreslený obsah, tím více unikátních informací lze získat, avšak za cenu častějších změn, delší doby zpracování a více potřebného místa pro uložení získaných dat. Velikost dat lze v praxi zmenšit pomocí hašovací funkce, ale za cenu další ztráty unikátnosti. [29]

[47] https://html.spec.whatwg.org/multipage/canvas.html

Získání všech dat z prvku canvas je možné pomocí funkcí[48] `toDataURL` a `toBlob`. Funkce `toDataURL` používá kódování Base64 pro reprezentaci obsahu a funkce `toBlob` umožňuje z obsahu vytvořit obrazový soubor. Alternativně lze získat i reprezentaci jednotlivých pixelů z 2D grafického kontextu pomocí funkce `getImageData`[49]. Pixely jsou reprezentovány rozhraním `ImageData`, který specifikuje formát dat RGBA (Red, Green, Blue, Alpha) pro každý pixel.

3.8.1 Vyhlazování písma

Vyhlazování písma (angl. font smoothing, font anti-aliasing) [53, 12] je technika, která má za cíl zlepšit čitelnost písma. V zobrazovacích zařízeních LCD jsou jednotlivé pixely tvořeny několika seřazenými primárními sub-pixely reprezentovanými typicky jako RGB (red-green-blue).

Písmo může být vykresleno bez jakéhokoliv vyhlazení. V takovém případě jsou jednotlivé pixely buď „zapnuty“, nebo „vypnuty“. V případě černého textu na bílem pozadí budou pixely pouze černé, nebo bílé. Zaoblené tvary písmen pak nemusí působit ostře, zejména na zobrazovacích zařízeních s nižším rozlišením. Techniky, které vyhlazují písmo, se dělí na:

- **Vykreslování ve stupních šedi** (angl. Grayscale rendering) – Pixely již nemusí být pouze „zapnuty“, nebo „vypnuty“, ale mohou nabývat i stupňů šedi. V případě černého textu na bílém pozadí mohou pixely nabývat hodnot v rozmezí 0–255.
- **Sub-pixelové vykreslování** (angl. Sub-pixel rendering) – Tato technika pracuje na úrovní sub-pixelů. U sub-pixelů je zvlášť kontrolován jas, a tím pádem může docházet k různým barvám pixelů, zejména na jejich okrajích. Tato technika sice zvyšuje rozlišení písma, ale zanechává nechtěné barevné artefakty.
- **Technologie ClearType** – Je technika implementovaná společností Microsoft, která vylepšuje nedostatky v podobě barevných artefaktů v sub-pixelovém vykreslování.

Detekce vyhlazování písma

Princip detekce vyhlazování písma spočívá ve vylepšení algoritmu [21] představeném již v roce 2009. Původní myšlenka algoritmu spočívá ve vykreslení černého písmena `"O"` na bílé pozadí prvku canvas. Následně je provedena analýza pixelů. Při vypnutém vyhlazování jsou všechny pixely černé, nebo bílé, ale při zapnutém vyhlazování písma jsou některé pixely průhledné. Rodina prohlížečů Trident navíc jako jediná implementuje nestandardní atribut `screen.fontSmoothingEnabled`[50], který indikuje zapnuté/vypnuté vyhlazování písma.

Vylepšení původního algoritmu, které navrhuji, spočívá v počítání průhledných pixelů. Výsledná informace tedy nebude pouze pravdivostní hodnota, ale číslo poskytující potenciálně vyšší míru informace. Další vylepšení může spočívat v podrobnější analýze průhledných pixelů, tedy pokud se hodnoty všech sub-pixelů rovnají, pak se jedná o vykreslování ve stupních šedi, jinak se jedná o sub-pixelové vykreslování neimplementující korekci barevných artefaktů.

[48] https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement
[49] https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/getImageData
[50] http://help.dottoro.com/ljnioacj.php

3.8.2 Vykreslování textu v písmu Arial

Mowery et al. [37] se jako první v roce 2012 zabýval otázkou získávání informací z prvku canvas. V prvku canvas byl vykreslen pangram[51] „How quickly daft jumping zebras vex.“ spolu s další interpunkcí v písmu Arial, které je staré již 39 let.

Z analýzy vyplynulo, že na základě různého vykreslení textu lze odhalit operační systém a téměř vždy také rodinu prohlížečů. Kromě operačního systému a prohlížeče může ovlivnit vykreslení textu také grafická karta, instalovaná písma a způsob vyhlazování písma.

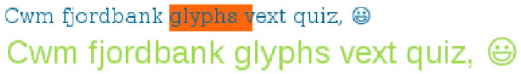
Kobusińska et al. [29] v roce 2017 zjistila, že přidání čísla k textu zvýšilo celkovou rozmanitost. Testovány byly i speciální znaky, avšak pouze jako rozšíření původní metody, takže z nich nebyly vyvozeny žádné závěry.

3.8.3 Vykreslení textu v záložním písmu

Vykreslení textu v záložním písmu [37] má za cíl vynutit použití záložního písma. Záložní písmo bude použito, pokud má prohlížeč za úkol vykreslit text pomocí neexistujícího nebo nedostupného písma. Záložní písmo se může lišit mezi prohlížeči i operačními systémy.

V roce 2014 Acar et al. [3] v rámci analýzy sledovacích skriptů našel použití technik vykreslení textu v záložním písmu a písmu Arial. Kromě těchto technik používal skript společnosti AddThis další techniky, jejichž výsledek lze pozorovat na obrázku 3.3.

(a) Obrázek vykreslený v prvku canvas skriptem společnosti AddThis, který analyzoval Acar et al. (převzato z [3]).



(b) Laperdrix et al. v roce 2016 vyšel z obrázku, který vykresloval skript AddThis. Jediná úprava spočívala ve změně pozic pangramů tak, aby se vzájemně nepřekrývaly (převzato z [30]).

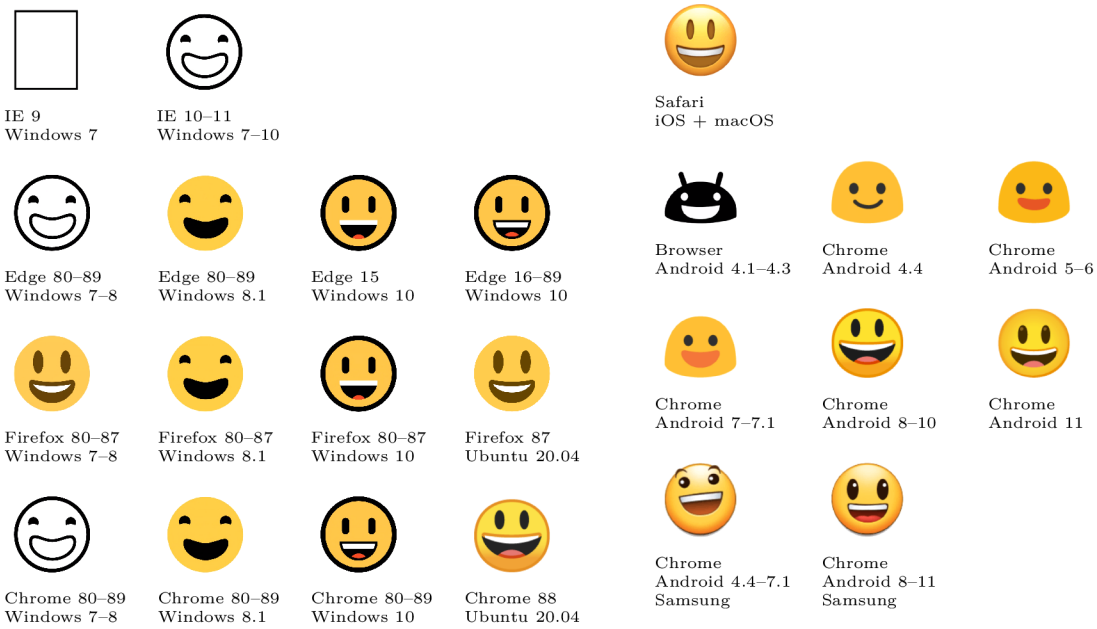
Obrázek 3.3: V prvku canvas je dvakrát vykreslený perfektní pangram pokaždé v jiné barvě. Jednou pomocí neexistujícího písma začínajícího názvem `"no-real-font-"` a podruhé pomocí písma Arial. Jedna z nových technik vykresluje znak Unicode `U+1F603`, který představuje „usměvavou tvář s otevřenými ústy“ (kapitola 3.8.4). Další technika testuje podporu algoritmů pro vyplňování oblastí pomocí funkce `isPointInPath` (kapitola 3.8.5).

3.8.4 Vykreslování emodži

Emodži (angl. emoji) [30] je ideogram znázorňující emoce nebo aktivity. Emodži jsou nástupci emotikonů, kde emotikony jsou složeny výhradně z posloupností písmen, číslic a interpunkčních znamének jako například „:)“ nebo „<3“. Každý emodži má přiřazený znak Unicode, který ho reprezentuje. Tvůrci písem musí pro každý znak Unicode reprezentující emodži poskytnout jejich korespondující vzhled ideogramu. Na obrázku 3.4 je zobrazeno rozdílné vykreslení emodži reprezentujícího „usměvavou tvář s otevřenými ústy“ na několika operačních systémech a prohlížečích.

Vzhledem k velké diverzitě různého vykreslení poskytuje usmívající se emodži velkou míru informace. Bylo však zjištěno, že způsob vykreslení emodži se oproti jiným informacím

[51] Pangram je věta nebo text obsahující všechna písmena abecedy alespoň jednou. Perfektní pangram obsahuje všechna písmena abecedy právě jednou.



Obrázek 3.4: Rozdílné vykreslení stejného emodži v různých rodinách prohlížečů a typech operačních systémů. Obrázek jsem vytvořil díky nástroji `browserstack.com` a mnou vytvořené webové stránce `emoji-canvas.hys.cz`.

z prvku canvas častěji mění [29]. Na základě změn ve vykreslení stejného emodži bylo možné rozpoznat aktualizace v operačním systému Windows 7 [34].

3.8.5 Vyplňování oblastí

Vyplňování oblastí [7] se zabývá nalezením všech vnitřních bodů dané oblasti v prvku canvas. Algoritmus pro určení, zda je bod uvnitř nebo mimo danou oblast, je definován vlastností `CanvasFillRule`, která nabývá jedné z hodnot `"nonzero"` a `"evenodd"`.

Funkce `isPointInPath`[52] akceptuje parametr typu `CanvasFillRule`. Tento parametr specifikuje algoritmus pro vyhodnocení, zda se bod nachází uvnitř nebo mimo oblast. Knihovna Modernizr[53] implementuje test podpory paritního vyplňování oblastí pomocí funkce `isPointInPath`. Informace o podpoře vyplňování oblastí zvyšuje míru získané informace [29].

3.8.6 Míchání barev

Mícháním různých barev [25] lze získat velké množství různých barevných odstínů. Barvy, ze kterých vznikají další odstíny, se jmenují základní (primární) barvy. Podle výběru základních barev se rozlišují dva základní druhy míchání barev:

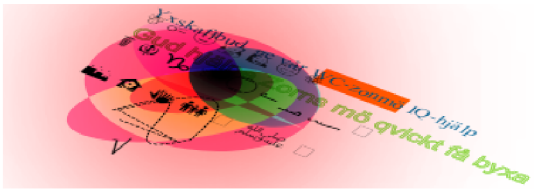
- **Aditivní míchání barev** – Základní barvy jsou RGB (červená, zelená, modrá), kde jejich složením vzniká bílá barva.
- **Subtraktivní míchání barev** – Základní barvy jsou CMY (azurová, purpurová, žlutá), kde jejich složením vzniká černá barva.

[52] `https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/isPointInPath`
[53] `https://github.com/Modernizr/Modernizr/blob/master/feature-detects/canvas/winding.js`

Míchání barev v prvku canvas je nastaveno atributem `globalCompositeOperation`, a pokud je to možné, využívá hardwarovou akceleraci z důvodu zvýšení výkonu [8]. Podpora míchání barev může být zjištěna pomocí úspěšného nastavení způsobu míchání, jak implementuje knihovna Modernizr[54]. Informace o podpoře míchání barev zvyšuje míru získané informace [29].

3.8.7 Další techniky získávání informací z prvku canvas

Gómez-Boix et al. [17] v roce 2018 vyšel z předešlých studií [37, 3], které se zabývaly prvkem canvas. Použil podobné techniky a přidal i některé nové. Veškeré aplikované techniky lze pozorovat na obrázku 3.5. Z výsledků studie vyplynulo, že informace získané z prvku canvas dosahují největší diverzity na mobilních zařízeních a druhé největší diverzity na osobních počítačích.



Obrázek 3.5: Obrázek demonstruje použité techniky pro získání informací. Mezi nové techniky patří: aplikace rotace na vykreslované objekty, vykreslení 4 matematických funkcí (sinus, kosinus a 2 lineární funkce), vykreslení několika elips s různou barvou, velikostí, úrovní průhlednosti a případně nastaveným barevným kruhovým přechodem (převzato z [17]).

3.8.8 WebGL

Technologie WebGL (Web Graphics Library) [37] definována specifikací[55] slouží k vykreslení 2D nebo 3D obsahu ve 3D prostoru. Rozhraní WebGL je odvozeno od rozhraní OpenGL a nabízí podobnou funkcionalitu vykreslení, ale v kontextu prvku canvas.

Rozhraní nabízí dvě hlavní části `vertex shader` a `fragment shader`, jejichž naprogramování je realizováno v jazyce GLSL (OpenGL Shading Language). Po úspěšné kompilaci jsou obě části propojeny a spuštěny přímo na grafické kartě.

Informace o grafické kartě

WebGL prostřednictvím rozšíření `WEBGL_debug_renderer_info`[56] zprostředkovává informace o grafické kartě pro účely ladění programů. Tyto informace jsou:

- **WebGL vendor** – Jméno výrobce grafické karty.
- **WebGL renderer** – Název modelu grafické karty.

Oba atributy však obsahují další informace o operačním systému a jeho prostředí. Například model obsahující řetězce `"ANGLE"` a `"Direct3D11"` indikuje operační systém Windows a řetězec `"OpenGL engine"` zase indikuje operační systém macOS [30]. Řetězec `"VMware"` indikuje, že prohlížeč je spuštěn ve virtuálním stroji [54].

[54] https://github.com/Modernizr/Modernizr/blob/master/feature-detects/canvas/blending.js
[55] https://www.khronos.org/registry/webgl/specs/latest
[56] https://developer.mozilla.org/en-US/docs/Web/API/WEBGL_debug_renderer_info

3.9 Další metody získávání informací

Tato kapitola popisuje metody použitelné pro získání dalších informací o prohlížeči a zařízení uživatele. Na základě popsaných metod lze s určitou pravděpodobností detekovat architekturu systému, operační systém nebo režim chránící proti získávání otisku v rodině prohlížečů Gecko.

3.9.1 Detekce architektury pomocí instrukční sady

Schwarz et al. [52] představil útok postranním kanálem využívající vlastnosti JIT (just-in-time) kompilace kódu v jazyce JavaScript. Útok je založen na dvou podobných kódech v jazyce JavaScript. Oba kódy provádí velké množství operací, kde jednotlivé operace mají mezi sebou datové závislosti, v pohyblivé řadové čárce. Jeden kód má však o jednu operaci více než kód druhý. Na 64bitové architektuře mohou být všechny hodnoty obou kódů uloženy v registrech, zatímco na 32bitové architektuře se všechny hodnoty delšího kódu nevejdou do menšího počtu registrů. Na 32bitové architektuře je v případě delšího kódu nutné využít zásobník pro dočasné uložení hodnot, což zvyšuje počet nezbytných instrukcí pro vykonání kódu, a tím pádem výrazně stoupá čas na potřebné vykonání celého kódu.

3.9.2 Matematické výpočty

Specifikace jazyka JavaScript [19] nedefinuje všechny výsledky mnoha matematických funkcí v celém oboru hodnot, ale definuje pouze některé výsledky z okrajových hodnot definičního oboru daných funkcí. Volba algoritmů pro aproximaci výsledků není definována v rámci specifikace jazyka JavaScript, ale je doporučeno, aby implementace využívala aproximační algoritmy definované standardem `IEEE 754-2019`.

Na základě menších analýz [44] testování výsledků pro specifické hodnoty argumentů vybraných matematických funkcí byl demonstrován potenciál rozlišit mezi 32bit a 64bit architekturou a také mezi operačními systémy. Mezi původně testovanými funkcemi včetně argumentů byly: `asinh(1)`, `acosh(1e300)`, `atanh(0.5)`, `expm1(1)`, `log1p(10)`, `sinh(1)`, `cosh(10)`, `tanh(1)`, `tan(-1e300)`. Pozdější analýza testovala funkci kosinus, kde nejlepších výsledků dosahovala volání: `cos(1e272)`, `cos(1e0)`, `cos(1e284)`, `cos(1e75)`.

3.9.3 Detekce snížení přesnosti časových údajů

Prohlížeč Tor Browser [45] ve výchozím nastavení snižuje přesnost časových údajů na 100 ms. Ke stejnému snížení přesnosti dochází i v rodině prohlížečů Gecko, pokud uživatel na adrese `about:config` povolí možnost `privacy.resistFingerprinting`. Pokud jsou tedy výsledky volání například těchto funkcí `new Date().getTime()`, `performance.now()` nebo `new Event("").timeStamp` dělitelné 100, pak velmi pravděpodobně dochází ke snížení přesnosti časových údajů.

Kapitola 4

Získávání informací o chování uživatele

Kapitola popisuje informace, které lze získat z prohlížeče při interakci uživatele s webovou stránkou. Interakce typicky probíhá pomocí vstupních zařízení, jako klávesnice, myš, nebo také pomocí dotyku. Informace získaná z chování uživatele vyvolána jeho interakcí s webovou stránkou spadá do kategorie dynamických biometrických vlastností. Dynamické biometrické vlastnosti jsou však zatíženy vlivem prostředí a rozpoložením uživatele.

- **Dynamika pohybu myši** [26] je ovlivněna programovým nastavením (citlivost myši, rozlišení obrazovky, akcelerace pohybu, ...), zatížením zařízení, podložkou myši nebo psychickým stavem uživatele (únava, roztržitost, zoufalost, ...).
- **Dynamika stisku kláves** [57] je ovlivněna typem použité klávesnice (fyzická, virtuální), nastaveným rozložením klávesnice v operačním systému nebo psychickým stavem uživatele. Při zaznamenávání stisku kláves je také nutné řešit možný únik citlivých informací, jako jsou přihlašovací údaje k účtům nebo čísla platebních karet.

Použití dynamických biometrických vlastností v oblasti získávání informací spočívá spíše v extrahování dalších, dříve neznámých informací o uživateli. Mezi takové informace lze zařadit například predikce, zda je uživatel pravák/levák, do jaké věkové skupiny patří nebo jaké má pohlaví.

4.1 Události vyvolané interakcí s webovou stránkou

V této části kapitoly jsou popsány události, které lze zachytit při interakci s webovou stránkou pomocí klávesnice, myši nebo dotyku. Zahrnuty jsou i nejdůležitější atributy přítomné na vyvolaných událostech.

4.1.1 Interakce při použití klávesnice

Použití klávesnice vyvolává události `keydown`, `keyup` a `keypress`, které lze zachytit [16].

- `keydown` – Událost je vyvolaná, když dojde ke stisku klávesy. Po celou dobu, co je klávesa stisknuta, je tato událost neustále vyvolávána. Událost obsahuje atribut `repeat`, který je při prvním vyvolání nastaven na hodnotu `false` a v následujících vyvoláních je nastaven na hodnotu `true`.

- `keypress` – Událost podobná události `keydown`, ale je vyvolána pouze tehdy, když dojde k vytvoření znaku. Událost je případně vyvolaná až po události `keydown`.
- `keyup` – Událost je vyvolána, když dojde k uvolnění stisknuté klávesy. Událost je vyvolaná až po událostech `keydown` a `keypress`.

Události vyvolané stiskem klávesy implementují rozhraní `KeyboardEvent`, které kromě atributu `repeat` obsahuje mnoho dalších atributů. Mezi skupinu atributů se řadí i případně stisknuté modifikační klávesy (ctrl, shift, alt, meta). Další důležité atributy jsou `key` a `code`:

- `key` – Hodnota atributu odpovídá hodnotě klávesy, která je ovlivněna rozložením klávesnice a může být ovlivněna například modifikační klávesou shift.
- `code` – Hodnota atributu odpovídá fyzicky stisknuté klávese a není ovlivněna nastaveným rozložením klávesnice ani modifikačními klávesami.

4.1.2 Interakce při použití myši

Použití myši může vyvolat 9 různých událostí implementujících rozhraní `MouseEvent` [16]. Pro získávání informací jsou však nejrelevantnější následující 3 události, které zároveň nelze vyvolat pomocí klávesnice:

- `mousemove` – Událost je vyvolána pokaždé, když dojde k posunu kurzoru.
- `mousedown` – Událost je vyvolána pokaždé, když dojde ke stisku tlačítka na myši. Událost není opakovaně vyvolávána po dobu stisku tlačítka jako v případě stisku klávesy u klávesnice.
- `mouseup` – Událost je vyvolána pokaždé, když dojde k uvolnění stisknutého tlačítka na myši.

Rozhraní `MouseEvent` obsahuje informace o stejných modifikačních klávesách jako rozhraní `KeyboardEvent`. Další důležité atributy specifické pro myš jsou:

- `buttons` – Hodnota atributu reprezentuje, jaká tlačítka byla na myši stisknuta během vyvolané události.
- `clientX` a `clientY` – Souřadnice x a y relativní k levé horní pozici okna prohlížeče, ve které je zobrazena webová stránka. Hodnota není ovlivněna úrovní posunu na stránce.

4.1.3 Interakce při použití dotyku

Události vyvolané dotykem lze zachytit pomocí rozhraní `TouchEvent` [51], které není podporováno[1] v rodině prohlížečů Trident, nebo pomocí novějšího rozhraní `PointerEvent` [32], které kromě dotyku umí zachytit a rozlišit mezi interakcí pomocí myši a stylusu. Na mobilních zařízeních, na kterých záleží nejvíce, je podpora lepší pro starší rozhraní `TouchEvent`. Na rozhraní `TouchEvent` lze zachytit následující události [16]:

- `touchstart` – Událost je vyvolána pokaždé, když se prst dotkne obrazovky bez ohledu na to, jestli už se nějaký prst obrazovky dotýká.

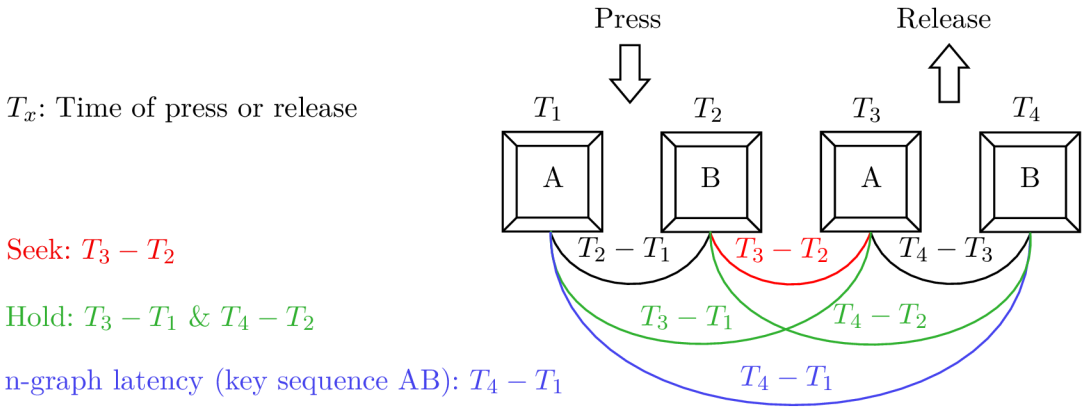
[1] https://caniuse.com/mdn-api_touchevent

- `touchmove` – Událost je vyvolána pokaždé, když se prst hýbe po obrazovce. Volání funkce `preventDefault` během této události zamezí posunu na stránce.
- `touchend` – Událost je vyvolána pokaždé, když se prst přestane dotýkat obrazovky.
- `touchcancel` – Událost je vyvolána pokaždé, když prohlížeč přestane detekovat dotyk. Taková situace může nastat, pokud uživatel přiloží příliš mnoho prstů na obrazovku.

4.2 Predikce preferované ruky pro psaní

Avar Pentel [42] dokázal predikovat s 99.5 % přesností pomocí klasifikační metody Random Forest, zda uživatel používá pravou, nebo levou ruku pro psaní na základě toho, jak píše na klávesnici. Studie bohužel nerozlišuje mezi úspěšností detekce na fyzických klávesnicích (notebook, externí klávesnice u stolního počítače) a virtuálních klávesnicích přítomných na mobilních zařízeních.

Data pro natrénování modelu strojového učení byla získána od 504 lidí prostřednictvím dotazníku. Sběr informací realizoval skript, který vytvářel trojici dat pomocí zachycení událostí `keydown` a `keyup` na objektu `window`. Trojice obsahovala hodnotu atributu `key`, čas stisku a uvolnění klávesy získaný funkcí `Date.getTime`. Vzhledem k použití atributu `key`, který je ovlivněn rozložením klávesnice, a relativně malého vzorku lidí vyvstává otázka efektivnosti modelu pro různé rozložení klávesnic. Skript neimplementoval žádné kódování znaků, takže by v praktickém nasazení mohlo docházet k nežádoucímu úniku citlivých informací. Základní extrahované rysy z těchto informací jsou popsány na obrázku 4.1.



Obrázek 4.1: Mezi základní extrahované rysy patří průměrná doba od stisknutí klávesy do doby jejího uvolnění (angl. hold), průměrná doba mezi posledním uvolněním klávesy a stiskem jiné klávesy (angl. seek) a průměrná doba zpoždění mezi specifickou sekvencí 2–4 kláves měřená od stisku první klávesy po uvolnění poslední klávesy z dané sekvence (angl. n-graph latency). Sekvence dvou napsaných písmen se nazývá digraf. (přepracováno z [42])

Pro získání výrazně lepších výsledků byly klávesy kategorizovány do dvou skupin podle toho, zda se fyzicky nachází na pravé nebo levé straně klávesnice. Na základě rozdělení na pravé a levé klávesy byly pro tyto skupiny zvlášť spočítány průměrné hodnoty rysů hold a seek. Z takto 4 spočítaných rysů byly spočítány další 2 rysy jako poměr levých a pravých korespondujících rysů hold a seek.

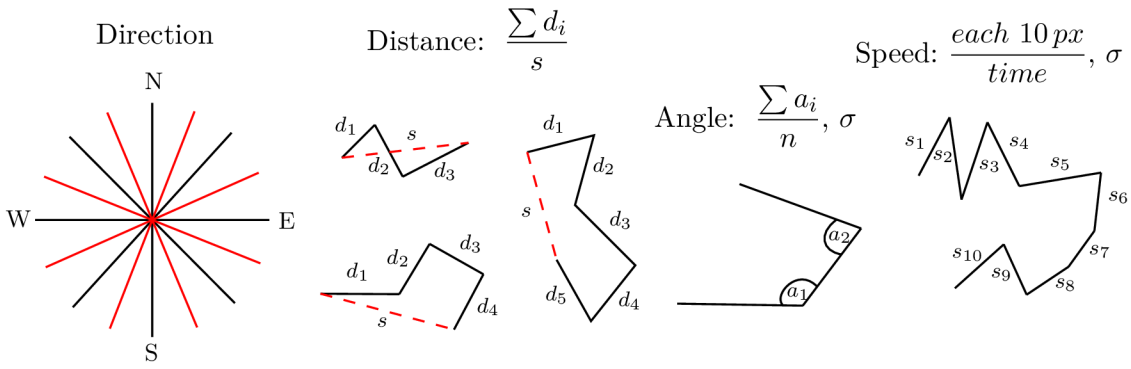
4.3 Predikce věkové skupiny a pohlaví

Avar Pentel [43] dokázal predikovat s více než 90 % přesností pomocí klasifikační metody Nearest Neighbor pohlaví nebo věkovou skupinu[2] uživatele na základě informací z pohybu myši. Pohlaví a věková skupina byla predikována i na základě informací z klávesnice, avšak pouze s přesností 73 %.

Data pro natrénování modelu strojového učení byla získána od 1 519 lidí během roků 2011–2017 při 6 různých příležitostech. V datové sadě se vyskytuje větší množství informací z pohybu myši než informací o stisku kláves, protože interakce s klávesnicí nebyla vždy vyžadována.

Základní rysy získané ze stisku kláves odpovídají rysům popsaným na obrázku 4.1. Navíc k nim byla přidaná směrodatná odchylka všech rysů, výskyt opravných kláves, poměr počtu napsaných znaků a celkové délky výsledného textu a průměrný čas rysů seek a hold pro všechny klávesy.

Sběr informací o pohybu myši realizoval skript, který zachytával události `mousemove` na objektu `document`. Vytvářeny byly opět trojice, kde trojice obsahovala dvojici souřadnic x, y a čas vytvoření trojice získaný funkcí `Date.getTime`. Zdroj souřadnic však nebyl přesně specifikován. Kromě dvojice souřadnic `clientX`, `clientY` mohou být totiž použity i dvojice souřadnic `screenX`, `screenY`, nebo `pageX`, `pageY`, které však mohou být ovlivněny pozicí okna prohlížeče, nebo úrovní posunu na stránce. Trojice informací byla vždy vytvořena, pokud došlo k posunu kurzoru o více než 10 px libovolným směrem. Extrahované rysy z této trojice informací jsou popsány na obrázku 4.2.



Obrázek 4.2: Mezi extrahované rysy patří zaznamenání počtu pohybů v určitých směrech, výpočet poměrů mezi délkou celé cesty a nejkratší možnou cestou, průměrný úhel, směrodatná odchylka všech úhlů a směrodatná odchylka spočítaných rychlostí za každých více než 10 px. (přepracováno z [43])

4.4 Predikce ruky držící mobilní zařízení

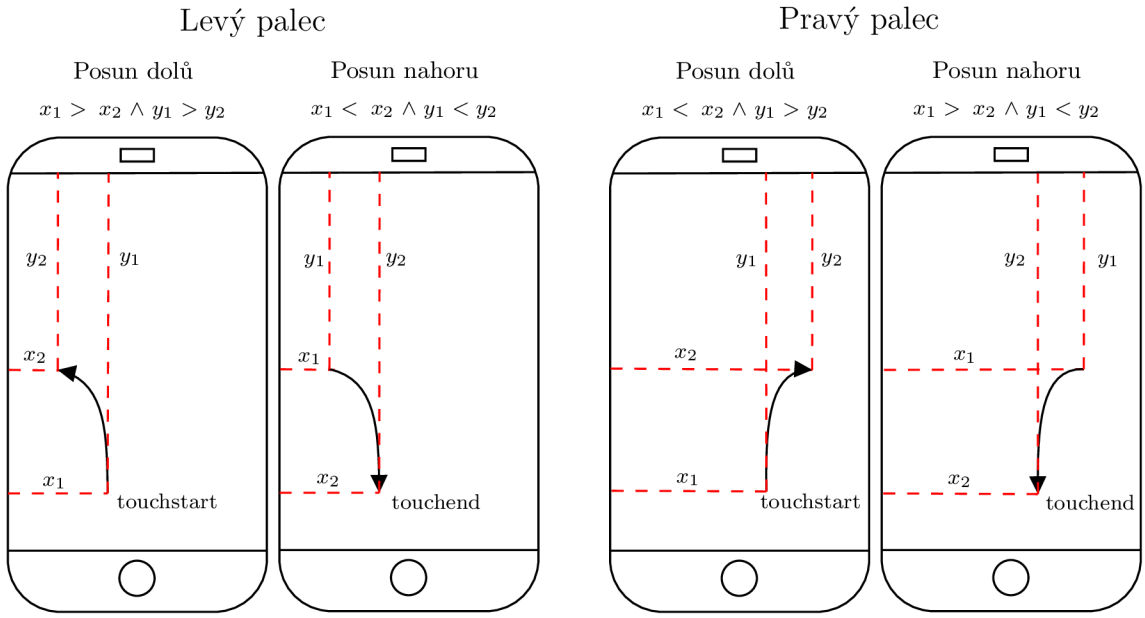
Freiberger [15] představil metodu, která predikuje, zda uživatel drží mobil v levé, nebo pravé ruce na základě pohybu palce při vertikálním posunu stránky směrem dolů. Data pro predikci jsou získány ze souřadnic prostřednictvím událostí `touchstart` a `touchend`.

[2] Věkové skupiny byly v této studii pouze 2. Jedna skupina byla 10–15 let a druhá 16 let a více.

Předpoklad pro úspěšnou predikci je, že pokud uživatel drží telefon v levé ruce, tak pro posun bude používat levý palce. Pokud takový uživatel posune stránku směrem dolů, pak se palec k levému okraje přiblíží, jinak se oddálí.

Autor v rámci článku uvažuje pouze posun stránky směrem dolů, při kterém je dostatečné porovnat pouze souřadnice na ose x. V praxi se však může stát, že uživatel přistoupí na stránku pomocí návěští, které ho odkáže jinam než na začátek, a pak může být jeho první akcí posun směrem nahoru. Pro detekci posunu směrem dolů i nahoru je nutné k porovnání přidat i souřadnice na ose y. Na obrázku 4.3 je znázorněna očekávaná dráha pohybu palce pro levou a pravou ruku v obou možných směrech posunu.

Zpřesnění detekce by mohlo být realizováno, pokud by se obrazovka logicky rozdělila na levou a pravou část, jak realizoval Avar Pentel v případě klávesnice (kapitola 4.2). Hypotéza je taková, že levák bude realizovat více pohybů blíže k levému okraji obrazovky a pravák zase blíže k pravému okraji obrazovky.



Obrázek 4.3: Detekce levé a pravé ruky pomocí pohybu palce při vertikálním posunu stránky.

4.5 Závěr

Analýza chování uživatele je velmi komplexní problematika, která může být ovlivněna velkou škálou různých faktorů. Mezi hlavní faktory se řadí prostředí a fyzický nebo psychický stav uživatele.

Praktické použití těchto informací pro vytvoření otisku uživatele je také komplikované z hlediska doby získání a přesnosti či důvěryhodnosti výsledků. Oproti informacím z kapitoly 3 je zisk těchto informací vždy podmíněn interakcí uživatele v jakékoliv podobě, která v nejhorším případě nemusí nikdy přijít, nebo nemusí přijít v dostatečné míře. Přesnost a důvěryhodnost těchto informací pro jednoho člověka může být jednoduše narušena. Pokud člověk bude chtít zmást algoritmy, které ho dříve správně kategorizovaly jako praváka ve středním věku na základě dynamiky pohybu myši a dynamiky stisku kláves, nic mu později

nebrání například v použití druhé ruky pro ovládání myši nebo použití virtuální klávesnice ovládanou myší, kterou může držet právě v druhé ruce.

Hlavní specifika při analýze interakce pomocí klávesnice spočívají v nutnosti předejít možnému úniku citlivých informací a rozlišení mezi fyzickými a virtuálními klávesnicemi. Riziko možného úniku informací lze omezit například vytvořením obecných skupin znaků podle rozložení klávesnice, čímž se samozřejmě sníží i míra získané informace. Analýza pohybu myši může být ovlivněna použitím touchpadu, jehož kvalita může mít rovněž vliv na získané informace.

Popsané studie [42, 43] používaly pro zjištění času při vytvoření záznamu vždy funkci `Date.getTime`. Ve skutečnosti však stačí přečíst atribut `timeStamp`[3], který je zděděn z rozhraní `Event` a který je přítomný na objektu vyvolané události. Hodnota tohoto atributu je v rodině prohlížečů Gecko celé číslo a v rodině prohlížečů Blink obsahuje i desetinná místa.

Tato práce se bude zabývat pouze návrhem a implementací získání nízkoúrovňových informací z vyvolaných událostí z popsaných interakcí. Podrobná analýza získaných nízkoúrovňových informací včetně zohlednění veškerých popsaných faktorů je však mimo rozsah této práce.

[3] https://developer.mozilla.org/en-US/docs/Web/API/Event/timeStamp

Kapitola 5

Získávání informací o webových rozšířeních

Webové rozšíření, která maskují identitu uživatele, provádí zásahy do prostředí webového prohlížeče. Tyto zásahy mohou způsobit vedlejší efekty, na základě kterých je možné určit, zda uživatel instaloval nějaké webové rozšíření do svého webového prohlížeče. Vedlejší efekty lze dělit podle typu získané informace na anomální a nekonzistentní.

Anomální informace je taková, jejíž přiřazená hodnota nemůže za normální situace nastat. Typickým zástupcem této kategorie je přepsání nativní funkce prohlížeče kódem třetí strany[1]. Nekonzistentní informace je taková, která má platnou hodnotu, avšak tato hodnota nemůže nastat současně s hodnotou jiné informace. Takovým příkladem může být situace, kdy informace z HTTP hlavičky `User-Agent` nekoresponduje s informací získanou z atributu `navigator.userAgent`.

Cílem této kapitoly je analyzovat a navrhnout pokročilejší techniky detekce, kterými lze odhalit anomální a nekonzistentní informace. Praktický přínos této kapitoly spočívá v tom, že na jejím základě bude možné určit:

- zda a případně jaké webové rozšíření uživatel nainstaloval do svého prohlížeče,
- jaké nekonzistentní a anomální informace vyřadit z tvorby otisku z důvodu narušení jejich důvěryhodnosti. Pokud například bude detekována anomálie ve funkci `toDataURL`, nebudou informace z prvku canvas zahrnuty v otisku.

5.1 Anomální informace

Anomální informace, na rozdíl od nekonzistentních informací, nelze získat pouze z informací popsaných v kapitole 3. Základní anomální informace lze získat z kontroly datového typu informace nebo ověření, zda přiřazená hodnota může za standardní situace nastat. Pokročilejší anomální informace, kterými se bude tato kapitola zabývat, je nutné získat novými technikami představenými v této kapitole.

5.1.1 Detekce falešné nativní funkce

Nativní funkce prohlížeče je taková funkce, která je implementována přímo prohlížečem a splňuje požadavky specifikace [19]. Za falešnou nativní funkce lze považovat takovou

[1] Třetí strana je v tomto okamžiku kdokoliv jiný než prohlížeč.

funkci, která se tváři jako nativní, ale je implementována třetí stranou. V kontextu maskování identity uživatele falešně nativní funkce například přidávají šum do výsledků, nebo úplně mění výsledky. Mezi falešné nativní funkce patří i takzvané funkce `polyfill`[2], jejichž výsledek operace by však měl být vždy totožný s výsledkem nativních funkcí.

Detekce falešné nativní funkce pomocí funkce `toString`

Každá funkce má dostupnou funkci `toString`, která pochází z prototypu objektu `Function` a která reprezentuje funkci jako řetězec. Algoritmus funkce `toString` od sebe odlišuje pomocí interních slotů 3 druhy funkcí. Druhy těchto funkcí jsou [19]:

- **Nativní funkce** – Funkce, která není implementovaná třetí stranou. Reprezentace takové funkce má syntaxi dle výpisu 5.1. Nativní funkce má vždy definované jméno v rámci slotu `[[InitialName]]` a je tedy vždy uvedeno. Formální list parametrů ve specifikaci funkce `toString` není nijak explicitně zmíněn a v praxi je pro nativní funkce vždy prázdný.
- **Běžná funkce** – Funkce, která má interní slot `[[SourceText]]`, který reprezentuje definici funkce. Jedná se o veškeré funkce pocházející od třetí strany, které současně nebyly vytvořeny pomocí funkce `Function.prototype.bind`. Reprezentace těchto funkcí je vždy závislá na konkrétní implementaci dané funkce. Běžné funkce vlastní interní sloty pro jméno funkce i formální list parametrů.
- **Exotická funkce** – Funkce, která nemá interní slot `[[SourceText]]`. Mezi tento typ funkcí se řadí funkce pocházející od třetí strany, které byly vytvořeny pomocí funkce `Function.prototype.bind`. Reprezentace exotických funkcí odpovídá reprezentaci nativní funkce. Exotické funkce neobsahují interní sloty pro jméno funkce ani formální list parametrů, tudíž tyto informace musí v reprezentaci nativní funkce chybět.

```
NativeFunction:
    function <functionName>(<formalParameters>) { [native code] }
```

Výpis 5.1: Zjednodušená reprezentace formátu nativní funkce.

Navržený algoritmus 1 pro každou validně definovanou funkci určí její druh. Druh funkce je určen na základě vlastností řetězcové reprezentace každé funkce.

Algoritmus 1: Pseudokód algoritmu detekce druhu funkce.

Vstup: Funkce
Výstup: Druh funkce
str_Function = získání řetězcové reprezentace funkce
norm_Function = odstranění komentářů a bílých znaků z proměnné str_Function
if *norm_Function tělo obsahuje pouze řetězec [nativecode]* **then**
  if *norm_Function signatura neobsahuje jméno funkce* **then**
    return "exotic"
  else
    return "native"
  end
else
  return "ordinary"
end

[2]Polyfill je kód, který implementuje webové API, které není v daném prohlížeči nebo jeho verzi dostupné.

Možné způsoby vytvoření falešné nativní funkce `toString`

Funkce `toString` slouží jako základní pravda o reprezentaci funkce v podobě řetězce. V případě, že by funkce byla zfalšována, další detekce na základě jejího výsledku již postrádají význam. Proto je nutné zjistit, zda byla zfalšována, a případně ji nahradit. Na základě místa zfalšování lze dělit zfalšování funkce `toString` na lokální a globální.

- **Lokální zfalšování** – Funkce `toString` bude implementována pouze na konkrétním objektu funkce a tím pádem zastíní funkci `toString` z objektu `Function`. Lokální zfalšování lze tedy vyřešit získáním řetězcové reprezentace funkce pomocí funkce `toString` z objektu `Function`.
- **Globální zfalšování** – Funkce `toString` bude zfalšována na objektu `Function`, ze kterého tuto funkci dědí všechny ostatní funkce, pokud současně nedochází k lokálnímu zfalšování. Funkce `toString` může být z prototypu objektu `Function` smazána a v takovém případě se použije funkce `toString` zděděná z objektu `Object`. Problematický případ nastává zejména, pokud dochází ke zfalšování takovým způsobem, že se falešná funkce `toString` tváří jako nativní, jak je uvedeno ve výpisu 5.2.

```
// Save native toString function.
const nativeToString = Function.prototype.toString;

// Replace native toString function with fake one, but keep original signature.
Function.prototype.toString = function toString() {
    const name = this.name || "";
    const binded = Function.prototype.bind(this);

    return nativeToString.call(binded).replace(/function /, "function " + name);
}
```

Výpis 5.2: Globální zfalšování funkce `toString` s dynamickou návratovou hodnotou.

Získání nefalešné nativní funkce

Při úspěšné detekci falešné nativní funkce nebo pouze z preventivních důvodů lze získat nefalešnou nativní funkci. Funkce je získána z nového vnořeného kontextu procházení, který v ideálním případě nebyl ovlivněn skriptem třetí strany. Takový kontext lze získat z nově vytvořeného prvku `iframe`[3] [13]. Následující výpis 5.3 demonstruje získání nefalešné nativní funkce `toString` z kontextu prvku `iframe`.

```
const iframe = document.createElement("iframe");
document.body.appendChild(iframe);
// Native toString function from iframe context, which can be used later on.
const iframeToString = iframe.contentWindow.window.Function.prototype.toString;
iframe.parentNode.removeChild(iframe);
```

Výpis 5.3: Získání nefalešné funkce `toString` z vnořeného kontextu prvku `iframe`.

Funkce `iframeToString` z výpisu 5.3 umožní získat správnou řetězcovou reprezentaci funkce `toString` z výpisu 5.2 voláním `iframeToString.call(Function.prototype.toString)`. Použití funkce `iframeToString` není samozřejmě limitováno pouze na funkci `toString`, ale lze s ní získat potenciálně správnou reprezentaci jakékoliv jiné funkce.

[3] https://developer.mozilla.org/en-US/docs/Web/API/HTMLIFrameElement/contentWindow

5.1.2 Detekce vlastnosti rozšiřitelnosti objektů

Každý objekt [19] v jazyce JavaScript obsahuje interní slot `[[Extensible]]`. Tento slot obsahuje pravdivostní hodnotu, která určuje, zda je možné přidávat nové atributy do objektu. Pokud není specifikováno jinak, nativní objekty jsou ve výchozím stavu rozšiřitelné a mají tedy nastavenou hodnotu slotu na `true`. Výjimkou jsou objekty, které jsou ve specifikaci definovány jako nerozšiřitelné, jako například `navigator.languages` [1]. Vlastnost rozšiřitelnosti objektů lze zkontrolovat voláním funkce `Object.isExtensible`. Volání funkce by za standardní situace, kromě výjimek, mělo skončit s výsledkem `true`.

5.1.3 Detekce rozdílných výsledků z více běhů funkcí

Princip této detekce spočívá minimálně ve dvou spuštění funkcí a porovnání jejich výsledků. Pokud výsledky nejsou stejné, je detekována anomálie. Tato detekce má za cíl detekovat zejména webové rozšíření, která přidávají šum náhodným způsobem.

5.1.4 Detekce možnosti smazání atributu z objektu

Nikiforakis et al. [40] podrobil objekty `navigator` a `screen` testu, ve kterém se je opakovaně pokoušel smazat včetně některých jejich atributů. Smazání vždy proběhlo úspěšně a žádná výjimka nebyla vyvolána. Smazání neproběhne úspěšně, pokud je objekt uzavřen vůči konfiguraci. Možnosti konfigurace definuje pravdivostní hodnota vnitřního slotu `[[Configurable]]`.

Smazání atributu na objektu je možné, pokud je objekt uzavřen vůči rozšíření funkcí `Object.preventExtensions`. Pokud je však na objektu zavolána funkce `Object.seal`, nebo `Object.freeze`, stává se objekt uzavřen vůči konfiguraci a smazání neproběhne úspěšně. Ve striktním režimu bude navíc vyvolána výjimka.

Smazání atributu přímo na objektu neovlivní atribut na prototypu a mělo by tedy být bezpečné smazat atribut přímo na objektu bez jakýchkoliv vedlejších efektů. Na základě tohoto předpokladu je tedy možné detekovat anomálii v konfigurovatelnosti objektu nebo v různých hodnotách atributu před a po jeho smazání.

5.2 Nekonzistentní informace

Nekonzistentní informace jsou detekovány ve více kategoriích, kde se každá kategorie sestává z několika zdrojů informací. Kategorie obsahuje nekonzistentní informace, pokud se všechny výsledky různých zdrojů informací v dané kategorii nerovnají. Tímto způsobem je možné detekovat nekonzistentní informace na úrovni kategorií, ale i na úrovni jednotlivých informací.

5.2.1 Rodina prohlížečů

Výsledkem zdrojů informací v kategorii rodiny prohlížečů je řetězec identifikující konkrétní rodinu prohlížeče. Kromě níže uvedených zdrojů informací lze také získat rodinu prohlížeče z atributu `navigator.userAgent`.

Výjimky

V jazyce JavaScript [19] existuje 7 standardizovaných typů výjimek a jedna nestandardizovaná výjimka `InternalError`[4]. Při vyvolání výjimky dojde k vytvoření objektu, který vždy obsahuje atributy `name` popisující typ výjimky a `message` popisující nastalou situaci.

Webové prohlížeče reagují na stejně vyvolanou chybu jiným typem výjimky nebo jiným popisem nastalé situace a přítomny mohou být i další nestandardní atributy na vytvořeném objektu [54]. Rozdílnou výjimku lze pozorovat v případě, kdy nebude ukončena rekurze, jak je demonstrováno v tabulce 5.1. Na základě těchto rozdílů je možné určit rodinu prohlížečů.

Zdroj vyvolání výjimky: **neukončená rekurze**			
Rodina prohlížečů	**`name`**	**`message`**	**Další atributy vyjma `name` a `message`**
WebKit	RangeError	Maximum call stack size exceeded.	line, column, stack
Blink		Maximum call stack size exceeded	stack
Gecko	InternalError	too much recursion	fileName, lineNumber, columnNumber
EdgeHTML	Error	Out of stack space	description, number, stack
Trident			–

Tabulka 5.1: Mapování rodiny prohlížečů na výsledky různých reakcí na stejně vyvolanou chybu. Na základě kombinace informací z pozorovaných kategoriích je mezi sebou možné odlišit všechny rodiny prohlížečů. Nestandardní atributy byly získány funkcí `Reflect.ownKeys`, která v rodině prohlížečů Trident není podporovaná.

Řetězcová reprezentace funkce `eval`

Vastel et al. [54] zjistil, že řetězcová reprezentace funkce `eval`, resp. délka řetězce dané reprezentace, se liší mezi rodinami prohlížečů vyjma rodin Gecko a WebKit, kde je délka rovna 37. Pro rozlišení těchto dvou rodin byl přidán 2. atribut `navigator.productSub`, který v kombinaci s délkou řetězcové reprezentace dokázal rodiny odlišit. Vastel et al. však do výzkumu nezahrnul rodinu EdgeHTML, která za aktuálního stavu není rozeznatelná od rodiny Blink.

Na základě výše zmíněného nedostatku navrhuji do detekce rodiny přidat 3. atribut `navigator.vendor`, na základě kterého je již možné od sebe rozlišit i rodiny Blink a EdgeHTML. Výslednou detekci rodiny prohlížečů na základě kombinace 3 hodnot lze pozorovat v tabulce 5.2.

Z tabulky 5.2 dále vyplývá, že pro detekci rodiny prohlížečů stačí pouze kombinace 2 hodnot. Konkrétně lze použit dvojice (`eval.toString().length`, `navigator.vendor`) a (`navigator.productSub`, `navigator.vendor`).

Detekce rodiny prohlížečů pomocí této metody může být za nestandardních situací méně spolehlivá než detekce založená na výjimkách (kapitola 5.2.1). Hlavní důvod spočívá v tom,

[4] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/InternalError

		navigator	
Rodina prohlížečů	`eval.toString().length`	`productSub`	`vendor`
WebKit	37	`"20030107"`	`"Apple Computer, Inc."`
Blink	33		`"Google Inc."`
EdgeHTML			`""`
Gecko	37	`"20100101"`	
Trident	39	`undefined`	

Tabulka 5.2: Detekce rodiny prohlížečů na základě kombinace 3 hodnot.

že je mnohem jednodušší ovlivnit statické hodnoty na objektu `navigator`, přepsat funkci `eval` nebo `toString`, než ovlivnit způsob vykonávání kódu.

Pluginy

V rodině prohlížečů Gecko je v jako jediné zakázaná[5] enumerace `navigator.plugins` a `navigator.mimeTypes`, čímž se od ostatních rodin odlišuje. V následujícím seznamu jsou zobrazeny rozdíly v pluginech v rámci různých rodin prohlížečů. Veškeré hodnoty byly získány enumerací na objektu `plugins` ve výchozím nastavení prohlížeče.

- **Blink** – Pluginy, resp. jejich názvy, se v této rodině prohlížečů liší napříč prohlížeči. Přítomnost či nepřítomnost pluginu má potenciál detekovat konkrétní typ prohlížeče.
  - **Google Chrome** – `"Chrome PDF Plugin"`, `"Chrome PDF Viewer"` a `"Native Client"`.
  - **Opera** – `"Chromium PDF Plugin"`, `"Chromium PDF Viewer"` a `"News feed handler"`.
  - **Edge** – `"Microsoft Edge PDF Plugin"`, `Microsoft Edge PDF Viewer"` a `"Native Client"`.
  - **Seznam.cz** – `"Chromium PDF Plugin"` a `"Chromium PDF Viewer"`.
  - **Brave** – Při každém novém otevření prohlížeče jsou vygenerovány jiné pluginy, kde jednotlivé hodnoty jsou často maskovány. Počet vygenerovaných pluginů byl vždy 4. Příklad zamaskovaného jména pluginu je `"069. fv"`.
- **WebKit** – Pluginy v této rodině se liší v majoritních verzích, čímž vzniká potenciál detekovat dokonce konkrétní majoritní verzi prohlížeče Safari.
  - **Safari 13.1** – `"Shockwave Flash"`.
  - **Safari 12.1** – `"WebKit built-in PDF"`.
  - **Safari 11.1** – `"Shockwave for Director"` a `"WebKit built-in PDF"`.
- **EdgeHTML** – Prohlížeč Edge obsahuje `"EDGE PDF Viewer"` a `"Shockwave Flash"`.
- **Trident** – V prohlížeči Internet Explorer 11 je přítomný pouze plugin s názvem `"Shockwave Flash"`. V nižších verzích 10 a 9 nejsou typicky pluginy obsaženy.

[5] https://developer.mozilla.org/en-US/docs/Web/API/NavigatorPlugins/plugins

V rodině prohlížečů Gecko a prohlížečích Internet Explorer 9 a 10 nelze iterací získat žádné pluginy. Pro odlišení stačí použít atribut `navigator.languages`, který v rodině prohlížečů Trident není podporovaný (kapitola 3.1.4).

5.2.2 Operační systém

Výsledkem zdrojů informací v kategorii operačních systémů je řetězec identifikující konkrétní operační systém. Kromě níže uvedených zdrojů informací lze také získat operační systém z atributu `navigator.userAgent`.

Platforma

Vastel et al. [54] publikoval tabulku, která mapuje běžné operační systémy na hodnoty z atributu `navigator.platform`. Tuto tabulku jsem převzal a rozšířil o několik hodnot z téhož atributu, které jsou odlišeny podtržením. Výsledek je zobrazen v tabulce 5.3.

Operační systém	**`navigator.platform`**
Windows	Win32, Win64, Win16, WinCE
Linux	Linux i686, Linux x86_64
macOS	MacIntel, Macintosh, MacPPC, Mac68K
iOS	iPhone, iPad, iPod
Android	Linux i686, Linux armv71, Linux armv81
FreeBSD	FreeBSD amd64, FreeBSD i386

Tabulka 5.3: Mapování různých hodnot platformy k danému operačnímu systému.

WebGL

Na základě informací `renderer` (model grafické karty) a `vendor` (výrobce grafické karty) poskytovaných rozhraním WebGL lze definovat mapování mezi kombinací těchto hodnot a korespondujícím operačním systémem. Mapování je zobrazeno v tabulce 5.4.

Operační systém	**`renderer`**	**`vendor`**
Windows	ANGLE	Microsoft, Google Inc
Linux	Mesa, Gallium	Intel, VMWare, X.Org
macOS	OpenGL, Iris	Intel, ATI
iOS	Apple, PowerVR	Apple, Imagination
Android	Adreno, Mali, PowerVR	Qualcomm, ARM, Imagination
Windows Phone	Qualcomm, Adreno	Microsoft

Tabulka 5.4: Mapování operačních systémů na řetězce vyskytující se v atributech `renderer` a `vendor` (převzato z [54]).

5.2.3 Zobrazovací zařízení

Výsledkem zdrojů informací v kategorii zobrazovacích zařízení je pravdivostní hodnota, kde hodnota `true` znamená, že byla detekována nekonzistentní informace v daném zdroji.

- **Orientace** – Porovnání šířky a výšky v závislosti na typu orientace.
- **Barevná hloubka** – Porovnání hodnot atributů `colorDepth` a `pixelDepth`.
- **Rozměry** – Hodnota atributu `availWidth` nemůže být větší než hodnoto atributu `width`. Analogicky to stejné platí i pro výšku.
- **Poměr rozměrů** – Porovnání poměrů atributů udávajících šířku a výšku. Poměr je nutné zaokrouhlit na jedno desetinné místo kvůli přítomným zaokrouhlovacím chybám.
- **Dotyk** – Nekonzistentní informace o podpoře dotyku je získána na základě kombinace hodnot z detekcí pomocí jazyka JavaScript a CSS (kapitola 3.3.3). Výsledek těchto detekcí musí být dále v souladu s hodnotou `maxTouchPoints`.

5.2.4 Preferované jazyky

Výsledkem zdrojů informací v kategorii preferovaných jazyků je pravdivostní hodnota. Detekovány jsou následující nekonzistentní informace.

- **Priorita preferovaného jazyka** – První jazyk v poli `navigator.languages` by měl odpovídat jazyku v atributu `navigator.language`.
- **Nestandardní atributy** – Rodina prohlížečů Trident implementuje nestandardní atributy demonstrující jazykové preference a zároveň neimplementuje standardní atributy. Nekonzistentní informace je detekována, pokud jsou přítomny standardní i nestandardní atributy současně.

5.2.5 Úložiště

Výsledkem zdrojů informací v kategorii úložišť je pravdivostní hodnota. Detekovány jsou následující nekonzistentní informace.

- **Webové úložiště** – Úložiště `sessionStorage` a `localStorage` jsou konfigurovány společně a tedy informace o podpoře by měla být totožná.
- **Cookies** – Informace o podpoře ukládání souborů cookies by měla korespondovat s hodnotou atributu `cookieEnabled`.

5.2.6 HTTP hlavičky

Výsledkem zdrojů informací v kategorii HTTP hlaviček jsou pravdivostní hodnoty. Porovnávány jsou vždy informace získané z HTTP hlaviček s informacemi získanými z jazyka JavaScript. Detekovány jsou následující nekonzistentní informace.

- **Řetězec `userAgent`** – Porovnány jsou zvláště informace o prohlížeči, rodině prohlížečů, zařízení, operačním systému a architektuře procesoru.
- **Preferované jazyky** – Porovnány jsou preferované jazyky v pořadí, ve kterém byly doručeny a v seřazeném pořadí.
- **Preference „Do Not Track“** – Porovnána je preference nesledovat, ve které jsou uvažovány i nestandardní atributy z rodiny prohlížečů Trident.

- **Client Hints** – Experimentální hlavičky Client Hints jsou určeny k porovnání informací o mobilním zařízení, platformě, architektuře procesoru, poměru pixelů a síťových informací.

5.3 Detekce webových rozšíření maskující identitu uživatele

Webová rozšíření maskující identitu uživatele mohou zakázat použití rozhraní, omezit počet jeho volání, nahradit původní hodnotu jinou hodnotou, nebo také přidat šum do výsledku.

- Zakázání použití rozhraní může být dále realizováno pro skripty první nebo třetí strany. Avšak tato strategie může narušit běh webových aplikací, takže není příliš uživatelsky přívětivá.
- Omezení počtu volání rozhraní by v budoucnu mohlo být implementováno i v prohlížečích, jak je zmíněno v Privacy Budget[6], který je součástí iniciativy Privacy Sandbox[7].
- Nahrazení původní hodnoty je nejjednodušší způsob, jak omezit identifikaci uživatele. Aby tato strategie byla důvěryhodná, je nezbytné zajistit, aby byly při změně současně ovlivněny i všechny další relevantní informace. Pokud bude nějaká informace opomenuta, vznikají nekonzistence, které lze detekovat. Současně je způsob, jakým dochází k nahrazení hodnoty, typicky implementovaný tak, že vytváří detekovatelné anomálie.
- Přidání šumu se používá zejména u komplexnějších zdrojů informací jako jsou písma, zvuk, prvek canvas nebo technologie WebGL. Šum může být přidaný uniformně v rámci jedné relace, nebo vždy zcela náhodným způsobem.

V tabulce 5.5 je zaznamenáno, jaké strategie webová rozšíření používají pro maskování uživatelovy identity a jaké anomální nebo nekonzistentní informace byly detekovány na základě představených technik.

5.3.1 Závěr

Ve všech 19 analyzovaných webových rozšířeních byly úspěšně detekovány nekonzistentní nebo anomální informace. Uživatelé těchto webových rozšíření mají sice maskované některé informace, avšak jejich maskování je zaplaceno vytvořením zcela nových informací, které lze taktéž detekovat. Dochází zde tedy k paradoxu, že webová rozšíření mající zvýšit anonymitu uživatelů vytváří nepřímo nové informace, které lze použít pro přesnější identifikaci uživatelů [31].

Velká část analyzovaných webových rozšíření taktéž způsobovala nekonzistence nebo anomálie různými způsoby, na základě kterých lze webová rozšíření od sebe potenciálně odlišit a detekovat. U rozšíření JavaScript Restrictor a Don't FingerPrint Me je velká část anomálií zastoupena falešnými nativními funkcemi. Z výsledků lze také pozorovat, že bylo detekováno 2× více anomálií než nekonzistencí, což může být zapříčiněno rozmanitými způsoby, kterými lze ovlivňovat informace, které následně mohou být nekonzistentní.

[6] https://github.com/bslassey/privacy-budget
[7] https://www.chromium.org/Home/chromium-privacy/privacy-sandbox

Webové rozšíření	Nahrazení hodnoty	Přidání šumu	Anomálie	Nekonzistence
AudioContext Fingerprint Defender		✓	1	0
Browser Fingerprint Protector	✓		8	2
Canvas Blocker – Fingerprint Protect	✓	✓	2	0
CanvasFingerprintBlock	✓		2	0
Canvas Fingerprint Defender	✓	✓	3	0
CyDec Platform Anti-Fingerprinting	✓		0	6
Don't FingerPrint Me	✓		34	0
Fingerprint Shield	✓		2	0
Fingerprint Spoofing	✓	✓	2	4
Font Fingerprint Defender		✓	1	0
Hide My Back	✓		0	2
JavaScript Restrictor (lvl. 3)	✓	✓	12	1
RandomUA	✓		0	4
Random User-Agent	✓		0	1
Resist Fingerprinting	✓	✓	1	3
Trace – Online Tracking Protection	✓		8	6
User-Agent Switcher	✓		0	4
User-Agent Switcher and Manager	✓		3	3
WebGL Fingerprint Defender		✓	1	1
Celkem: 19 rozšíření	16	8	80	37

Tabulka 5.5: Tabulka zobrazuje, jaké strategie maskování informací webová rozšíření používají a jaké informace lze z těchto rozšíření získat. Každé rozšíření bylo instalováno zvlášť, a to v prohlížeči Chrome nebo Firefox na systému Ubuntu. Získávání informací bylo realizováno na prázdné webové stránce bez použití vnořeného kontextu procházení a webová stránka používala protokol HTTPS, aby bylo možné použít veškeré informace. Určení strategie bylo realizováno na základě dokumentace rozšíření nebo zdrojového kódu. Počet anomálií udává celkový součet anomálií ve všech kategoriích. Počet nekonzistencí je součet počtu všech zdrojů informací s pravdivostním výsledkem `true` a počtu kategorií, kde se všechny výsledné řetězce nerovnají.

Kapitola 6

Analýza knihoven pro získávání informací

Kapitola obsahuje analýzu 4 knihoven, které získávají informace o zařízení a prohlížeči uživatele (kapitola 3). Analýza nezahrnuje knihovny získávající informace o chování uživatele (kapitola 4), ani knihovny zabývající se detekcemi nekonzistencí a anomálií (kapitola 5), protože, dle mého průzkumu v době psání práce, neexistují.

Pořadí analyzovaných knihoven určuje jejich popularita, která je odvozena od počtu týdenního stažení[1]. Analýza bude u každé knihovny vyhodnocovat zejména kvalitu jednotlivých získávaných informací na úrovni kódu. Přitom bude hodnocena i robustnost těchto informací vůči možnému ovlivnění těchto hodnot pomocí webových rozšíření.

6.1 FingerprintJS

Knihovna FingerprintJS[2], která vznikla v roce 2015, dosahující 20 000 týdenních stažení je nejrozšířenější knihovnou na získávání informací. Knihovna je spravována společností, která kromě veřejně dostupné verze nabízí i placenou verzi knihovny, která umožňuje získání více informací, a také server pro vyhodnocení zaslaných informací. Tato práce se bude zabývat veřejně dostupnou verzí knihovny, která je v době psání práce `3.0.5`. Jelikož se jedná o nejrozšířenější knihovnu, zaměřím se zejména na popis jejich objektivních nedostatků.

6.1.1 Počet logických procesorů

Při získaní informace o počtu logických procesorů není rozlišeno mezi případem, kdy atribut není dostupný (rodina prohlížečů Trident a staré verze prohlížečů), a případem, kdy počet nemohl být získán (dle specifikace [1] má být vráceno číslo 1). V obou případech je vráceno číslo 1, a proto nelze tyto dva případy od sebe rozeznat.

6.1.2 Webové úložiště

Při detekci webových úložišť `sessionStorage` a `localStorage` je ošetřen případ možného vzniku výjimky z důvodu jejich zakázání. Avšak pokud nastane výjimka, tak je o ní informace ztracena, protože je vrácena hodnota `true` stejně jako v případě, kdy úložiště není zakázané a současně je podporované. Vzhledem k tomu, že informace o podpoře je

[1] Počet týdenních stažení je dostupný na stránkách `https://www.npmjs.com` u každé knihovny.
[2] `https://www.npmjs.com/package/@fingerprintjs/fingerprintjs`

v obou případech přetypovaná na pravdivostní hodnotu, je ztracena i informace o zakázaní webových úložišť pouze pomocí konfigurace `dom.storage.enabled` v prohlížeči Firefox.

6.1.3 Informace o zobrazovacím zařízení

Z objektu `screen` je získáván atribut `colorDepth`, dvojice (`width`, `height`) a (`availWidth`, `availHeight`). Sběr dvojic je proveden tak, že každá dvojice atributů je vložena do pole, seřazena funkcí `sort` a následně je v poli prohozeno pořadí rozměrů funkcí `reverse`. Problém však nastane v případě, že monitor zobrazující obsah na výšku s rozměry (1080, 1920) bude po provedení operací `sort` a `reverse` nerozeznatelný od monitoru, který zobrazuje obsah na šířku a který dosahuje rozměrů před i po provedením operací (1920, 1080).

Operace řazení a prohození rozměrů, které autoři aplikovali, má pravděpodobně za účel zvýšit stabilitu získaných rozměrů, zejména pak na mobilních zařízeních, kde uživatel může jednoduše měnit orientaci zobrazení a tím pádem měnit získané rozměry.

Lepší řešení však spočívá ve využití informace z atributu `screen.orientation.type`, která reprezentuje aktuální orientaci zařízení. Na základě získané informace lze vyhodnotit odpovídající hodnoty atributů a není tak nutné provádět operace řazení a prohození, které zmenšují míru získané informace, a zároveň se zachová stabilita získané informace.

6.1.4 Časové pásmo

Časové pásmo je detekováno správně pomocí funkce `getTimezoneOffset`, avšak není plně využit potenciál možné získané informace. Autoři uvažují vliv letního času, čímž zvyšují stabilitu získané informace, ale místo detekce letního času nebo vrácení dvou potenciálně různých hodnot časových pásem, je vrácena pouze jedna hodnota. Vrácená hodnota je určena jako maximum z hodnot dvou časových pásem, které jsou uvedeny ve výpisku 3.2.

6.1.5 Písma

Detekce instalovaných písem se snaží najít 52 písem a využívá výhradně jazyk JavaScript. Nastavená velikost písma pro detekci je 48 px, což je v průměru o 23 px méně než bylo nastavené v představených studiích zabývajících se detekcí písma. Na vytvářený prvek `span` je aplikována řada CSS stylů pro odstínění prvku od případných externích vlivů, které by mohly mít vliv na měření velikosti prvku, a tudíž na správnost detekce. Jako text prvku `span` byl zvolen řetězec `"mmMwWLliI0O&1"`. Změřené rozměry prvku `span` jsou poté porovnány vůči rozměrům pocházejícím z již dříve představených 3 výchozích písem, a pokud se rozměry alespoň v jednom případě liší, bylo písmo detekováno, jinak nikoliv.

6.1.6 Zvuk

Získání informací o zpracování zvukového signálu je realizováno prostřednictvím rozhraní `OfflineAudioContext`. Jádro algoritmu získávání informací je převzato ze studie [11], která techniku jako první zveřejnila. Do původního algoritmu bylo komunitou implementováno vylepšení[3], které řeší krajní situace, ve kterých nedocházelo ke zpracování signálu.

Pokud není rozhraní podporováno prohlížečem, je vrácena hodnota `-2` místo standardní hodnoty `undefined` reprezentující absenci rozhraní.

[3] https://github.com/fingerprintjs/fingerprintjs/pull/581

6.1.7 Prvek canvas

Získání informací pomocí prvku canvas využívá některých z dříve představených technik. Výsledky jednotlivě aplikovaných techniky skriptem FingerprintJS jsou zobrazeny na obrázku 6.1. Výsledná informace je však získána funkcí `toDataURL` až po aplikaci všech technik, jejichž výsledek je zobrazen na obrázku 6.2. Šířka prvku canvas je nastavena na 240 px a výška na 140 px.

(a) Vykreslení obdélníku bez aplikovaných transformací jako rotace nebo posun.

(b) Vykreslení emodži a textu pomocí záložního písma a písma Arial.

(c) Paritní vyplňování oblasti.

(d) Subtraktivní míchání barev.

Obrázek 6.1: Techniky použité skriptem FingerprintJS.

Obrázek 6.2: Výsledný obrázek po aplikaci všech technik skriptem FingerprintJS, který je použit pro získání informací.

6.1.8 Detekce nekonzistencí a anomálií

Knihovna ve 2. verzi obsahovala funkce `getHasLiedLanguages`, `getHasLiedResolution`, `getHasLiedOs` a `getHasLiedBrowser`, které již nyní neobsahuje a které dokázaly detekovat jednoduché nekonzistence v preferovaných jazycích, rozlišení, operačním systému a typu prohlížeče.

V některých případech jsou „napravovány“ datové typy získaných informací. Pokud se očekává, že hodnota má být například datového typu číslo, tak vždy dojde k přetypování na datový typ číslo. Tato náprava v podobě přetypování je implementována u informací `navigator.deviceMemory`, `navigator.hardwareConcurrency`, `Date.getTimezoneOffset` a informací udávajících rozměry na objektu `screen`.

6.1.9 Závěr

V rámci analýzy knihovny FingerprintJS byly popsány zejména zdroje informací, které vykazují nedostatky, nebo zdroje informací, které jsou komplikovanějšího charakteru.

Detekce nekonzistencí a anomálií není aktuálně ve 3. verzi knihovny nijak implementována. Naopak dochází ke ztrátě informací o nekonzistencích a anomáliích tím, že dochází k nápravě, resp. přetypování hodnot získávaných informací.

Knihovna nyní nabízí celkem 27 zdrojů informací. S příchodem 3. verze také skončila podpora rodiny prohlížečů Trident a obecně starších typu prohlížečů, které nepodporují rozhraní `Promise`.

6.2 ClientJS

Knihovna ClientJS[4] je aktuálně ve verzi `0.1.5` a vznikla 4 měsíce po prvním vydání knihovny FingerprintJS. Oproti FingerprintJS již není aktivně udržována a nabízí pouze 15 zdrojů informací, ze kterých se počítá otisk. Knihovna ve skutečnosti nabízí více informací, které ale nejsou zahrnuty v otisku a které jsou dostupné díky knihovně UAParser.js[5], která analyzuje řetězec `userAgent`. Z hlediska analýzy informací jsou zajímavé zejména následující zdroje informací:

- **Webové úložiště** – Kód totožný s kódem knihovny FingerprintJS včetně zmíněných nedostatků.
- **Časové pásmo** – Detekce časového pásma neuvažuje vliv letního času.
- **Písma** – Pro detekci se využívá kód již aktuálně neexistující knihovny, který je však v principu velmi podobný kódu knihovny FingerprintJS. Rozdíl je ve zvoleném řetězci `"mmmmmmmmmmlli"`, velikosti písma 72 px a celkem detekovaných 230 písmech.

6.3 Get Browser Fingerprint

Knihovna Get Browser Fingerprint[6] je aktuálně ve verzi `1.1.0` a byla vydána koncem roku 2019. Knihovna je velmi malá, implementovaná na 121 řádcích, a obsahuje 14 zdrojů informací. Na knihovně je zejména zajímavé, že nabízí vytvoření otisku pouze z informací týkajících se hardwaru zařízení. Do hardwarových informací jsou však zahrnuty i atributy `availWidth` a `availHeight` z objektu `screen`, které mohou být velmi silně ovlivněny konfigurací v operačním systému. Prvek canvas nevyužívá žádných nových technik, ale zvolil poměrně komplexní řetězec, který je uveden ve výpisku 6.1.

```
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
    `~1!2@3#4$5%6^7&8*9(0)-_=+[{]}|;:',<.>/?"
```

Výpis 6.1: Vykreslený řetězec knihovnou Get Browser Fingerprint v prvku canvas.

6.4 Simple Fingerprint

Knihovna Simple Fingerprint[7] je aktuálně ve verzi `1.1.0` a byla vydaná v roce 2020. Otisk je vytvořen z 10 zdrojů informací, které jsou označeny jako stabilní. Za nestabilní zdroje informací, které jsou poskytovány, ale nejsou zahrnuty do otisku, jsou považovány informace o stavu baterie. Z hlediska analýzy stabilních zdrojů informací se knihovna od ostatních odlišuje v následujících zdrojích informací:

- **Písma** – Knihovna využívá jazyka JavaScript pro detekci písem, ale pro jejich detekci využívá experimentální funkci `document.fonts.check`. Dle mých testů funguje detekce v prohlížeči Chrome, ale v prohlížečích Firefox a Safari jsou detekovány i neexistující názvy písem.

[4] https://www.npmjs.com/package/clientjs
[5] https://github.com/faisalman/ua-parser-js
[6] https://www.npmjs.com/package/get-browser-fingerprint
[7] https://www.npmjs.com/package/simple-fingerprint

- **Hlavičky protokolu HTTP** – Hlavičky jsou získány voláním REST API na adrese `https://httpbin.org/headers`.
- **Prvek canvas** – Na jeden z vykreslených textů je aplikován barevný lineární přechod.
- **WebGL** – Knihovna získává ladící informace z atributů `vendor` a `renderer`. Dále je pomocí WebGL vykreslena jednoduchá scéna v prvku canvas, ze které je následně získána informace funkcí `toDataURL`.

6.5 Závěr

Knihovnu FingerprintJS lze považovat za nejvyzrálejší knihovnu pro získávání informací. Je nejstarší, prošla nejvíce aktualizacemi a nabízí nejvíce zdrojů informací. Komplexnější techniky získávání informací jako písma, prvek canvas a zvuk vychází z řady studií. Z analýzy také vyplynulo, že aktuální implementace obsahuje několik nedostatků, pro které však bylo nastíněno řešení.

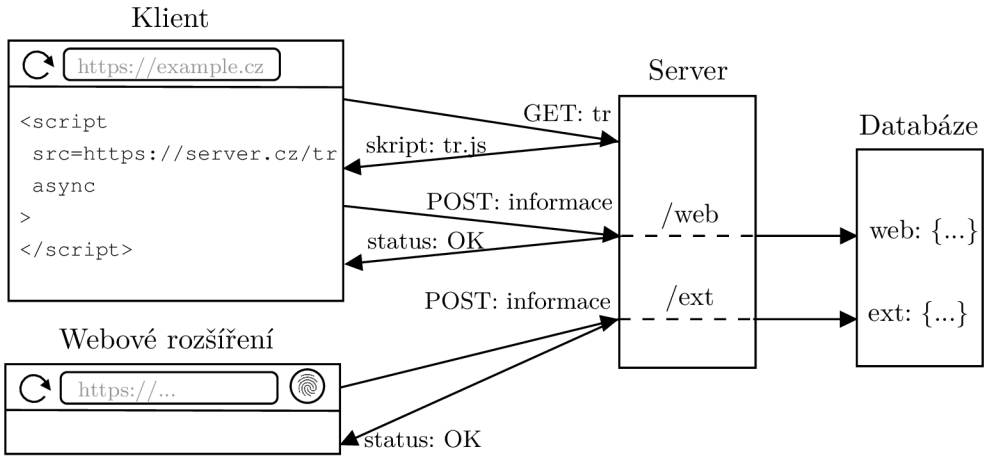
Z analýzy ostatních knihoven lze využít informace o existenci knihovny UAParser.js, komplexním řetězci zobrazeném ve výpisku 6.1 a implementaci vykreslení jednoduché scény pomocí WebGL.

Na základě analýzy lze také konstatovat, že žádná z knihoven v aktuální verzi neimplementuje techniky, které by detekovaly nekonzistence nebo anomálie. Z čehož plyne, že jakákoliv vytvořená nekonzistence nebo anomálie změní otisk takovým způsobem, že ho nebude možné spárovat s předchozím uživatelovým otiskem, protože nebude známo, jaké informace musí být vyřazeny z otisku. Současně všechny knihovny používají hašovací funkci pro vytvoření výsledného otisku.

Kapitola 7

Návrh systému pro získávání informací

Systém pro získávání informací je založen na architektuře klient-server. Na straně klienta je spuštěn skript, který získává informace popsané v předešlých kapitolách a který tyto informace zasílá skrze protokol HTTPS na server. Server poskytuje skript, který je spuštěn na straně klienta a zároveň ukládá přijatá data do databáze. V rámci práce bude navrženo i webové rozšíření. Webové rozšíření bude sbírat data od přesně identifikovaných uživatelů po delší časový úsek s cílem analyzovat stabilitu informací. Schéma fungování systému je zobrazeno na obrázku 7.1. Tato kapitola popisuje návrh jednotlivých části systému, kterými jsou knihovna pro získávání informací, webové rozšíření, server a databáze.



Obrázek 7.1: Schéma zachycuje klíčové propojení a komunikaci mezi částmi systému.

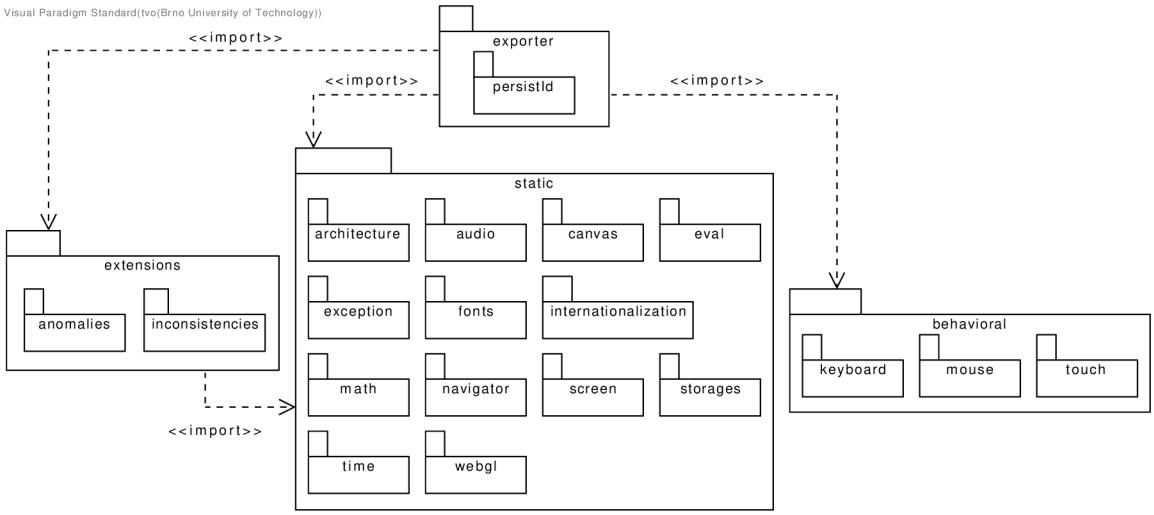
7.1 Knihovna pro získávání informací

Jedním z hlavních cílů práce je návrh a implementace knihovny pro získávání informací prostřednictvím webového prohlížeče. Při návrhu knihovny bude kladen důraz zejména na bezpečnost z pohledu klienta a na kvalitu výsledného kódu, aby bylo možné skript v praxi využít. Výsledný skript, který si klient může vyžádat ze serveru, představuje výsledek kompilace navržené knihovny.

7.1.1 Architektura knihovny

Architektura knihovny je navržena podle principu rozdělení balíčku na základě jejich vlastností (angl. package by feature) [20]. Podle tohoto principu balíčky obsahují veškerý nezbytný kód a snaží se co nejvíce se izolovat od ostatních balíčků. Na obrázku 7.2 je vytvořen diagram balíčků znázorňující nejdůležitější vazby v rámci knihovny. Největší výhody tohoto přístupu rozdělení, které v tomto případě spatřuji, jsou pro uživatele knihovny následující:

- Veškeré informace o daném zdroji informace se nachází na jednom místě a není nutné je vyhledávat v rámci celé knihovny.
- Vzhledem k izolaci jednotlivých balíčků je možné do cílové aplikace začlenit pouze nezbytnou část kódu knihovny. Tímto dochází ke snížení počtu přenášených dat po síti a také ke snížení zátěže na uživatelově zařízení, protože kód nemusí být přeložen.



Obrázek 7.2: Diagram balíčků knihovny, ve kterém jednotlivé balíčky získávají informace popsané v předešlých kapitolách. Výjimkou je balíček `exporter`, jehož zodpovědností je bezpečně odeslat získané informace z ostatních balíčků na server.

7.1.2 Statické informace

Statické informace, obsažené v balíčku `static`, jsou veškeré informace popsané v kapitole 3, které lze získat ihned po načtení webové stránky bez interakce uživatele. Získání statických informací je velmi často záležitostí přečtení hodnoty atributu. Ale i přečtení hodnoty atributu může být realizováno nevhodným způsobem, jak bylo zjištěno při analýze knihoven v kapitole 6. Největší identifikovaný problém nejpopulárnější knihovny FingerprintJS spočívá v tom, že u získaných informací zároveň provádí korekce pro zvýšení jejich stability, ale za cenu ztráty informační hodnoty. Z tohoto důvodu bude získání statických informací provedeno bez jakýchkoliv korekcí, aby byla zachována co možná největší informační hodnota. V rámci návrhu získání statických informací budou navržena vylepšení dosavadních způsobů získávání informací.

Nepodporované informace v prohlížeči

Při přistoupení k neexistujícímu atributu na objektu je za standardní situaci vrácena hodnota `undefined`. Tato hodnota tedy symbolizuje, že atribut není v daném prohlížeči nebo v jeho verzi podporován. Knihovna implementována v této práci vždy vrátí standardně očekávanou hodnotu `undefined` v případě, že atribut není podporován. Díky tomuto přístupu bude eliminována ztráta informací popsaná u knihovny FingerprintJS v kapitole 6.1.1.

Úložiště prohlížeče

V tabulce 3.1 bylo demonstrováno, že přístup k úložištím lze v prohlížeči různě konfigurovat. Konfigurace úložiště vždy ovlivní získanou hodnotu a může pomoci lépe identifikovat uživatele. Možné získané hodnoty jsou:

- **`undefined`** – Úložiště není podporováno.
- **`null`** – Úložiště bylo zakázáno na adrese `about:config` v prohlížeči Firefox.
- **`true`** – Úložiště je podporováno a není zakázáno.
- **`výjimka`** – Úložiště bylo zakázáno v rozhraní prohlížeče spolu se soubory cookies.

Pro získání maximální možné informace lze informaci o podpoře úložiště získat, jak je demonstrováno ve výpisku 7.1.

```
try {
    return <object>.<storage> && // undefined (not supported) or null (restricted)
           !!<object>.<storage>  // true (supported)
} catch(e) {
    return e.name;  // exception name (restricted)
}
```

Výpis 7.1: Detekce podpory úložiště s maximální mírou získané informace.

Prvek canvas

Prvek canvas představuje důležitý zdroj informací, které lze o uživateli získat, avšak ne všechny informace jsou v čase stejně stabilní. Významným zdrojem jsou emodži (kapitola 3.8.4), které se však velmi často mění. Z tohoto důvodu budou všechny techniky získávání informací z prvku canvas spuštěny nezávisle na sobě a nakonec i všechny společně.

Tímto přístupem tedy bude z prvku canvas extrahováno několik izolovaných informací a jedna informace kombinující všechny izolované informace. Výhoda izolovaných informací spočívá v tom, že pokud se změní výstup jedné informace, např. dojde k aktualizaci výchozího písma, nebudou tím ovlivněny ostatní informace a uživatel bude moci být s určitou pravděpodobností identifikován na základě výsledků ze zbylých informací.

Mezi techniky získání informací, které budou použity, patří: vyhlazování písma, vykreslení textu v písmu Arial, vykreslení textu v záložním písmu, vykreslování emodži, vyplňování oblastí, míchání barev, vykreslení barevného lineárního přechodu, vykreslení matematických funkcí a aplikace transformací na vykreslené objekty.

7.1.3 Behaviorální informace

Behaviorální informace, obsažené v balíčku `behavioral`, jsou veškeré informace popsané v kapitole 4, které lze získat skrze interakci uživatele s webovou stránkou. Interakce může probíhat pomocí klávesnice, myši nebo dotyku. Získání je založeno na přidání zpětného volání na kýžené události, kde zpětné volání je provoláno vždy, když dojde k interakci. Vzhledem k možnému počtu interakcí a k nutnosti zaslat získané informace na server, je nezbytné optimalizovat množství požadavků na server. Optimalizace počtu požadavků lze realizovat pomocí techniky zpracování mikro-dávek (angl. micro-batch processing) a pomocí vzorkování získaných dat.

Zpracování mikro-dávek

Technika zpracování mikro-dávek [33] funguje na principu akumulace většího počtu dat předtím, než jsou data odeslána na server. Odeslání, resp. zpracování dat na serveru však probíhá téměř v reálném čase, protože data jsou akumulována v řádech stovek milisekund. Dávky dat jsou odeslány na server ke zpracování v jedné z těchto situací:

- Počet akumulovaných dat přesáhl určitý práh definující maximální velikost dávky, kterou je žádoucí zpracovat.
- Byla akumulována data, jejich počet sice nepřesáhl práh, ale už uběhla příliš dlouhá doba od poslední interakce a data je nutné zpracovat na serveru.

Vzorkování dat z pohybu myši

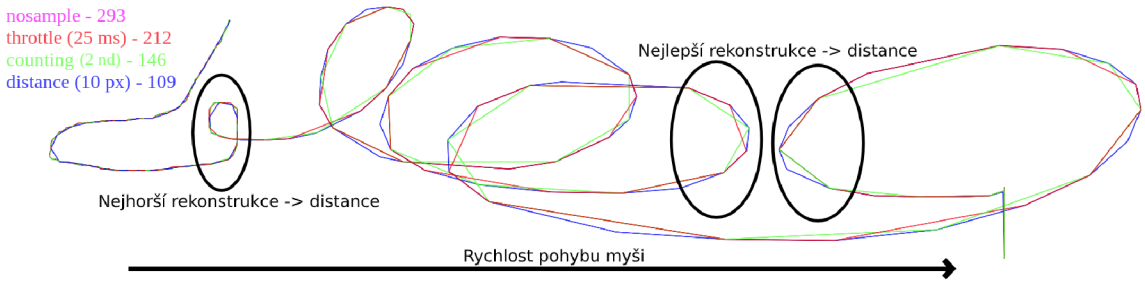
Vzorkování dat z pohybu myši může velmi omezit počet přenášených a analyzovaných dat. Při výběru vzorkovací techniky je nutné klást důraz na možnost spolehlivé rekonstrukce pohybu ze vzorkovaných dat. Realizoval jsem proto experiment, ve kterém jsem zvolil 3 vzorkovací techniky a pozoroval, jak přesně lze ze vzorkovaných dat rekonstruovat původní dráha pohybu myši. Současně bylo zaznamenáváno, kolikrát byl vzorek pohybu v rámci dané techniky získán. Použité vzorkovací techniky jsou:

- **Technika throttle**[1] – Vzorek pohybu je získán vždy maximálně jednou za uplynulý čas.
- **Technika počítání** (angl. counting) – Získán je každý n-tý vzorek pohybu.
- **Technika vzdálenosti** (angl. distance) – Tuto techniku představil Avar Pentel [43] a vzorek je získán vždy, když se kurzor posune o více než 10 px.

Demonstrace průběhu experimentu lze pozorovat na obrázku 7.3. V rámci experimentu jsem zvyšoval a snižoval hodnoty techniky throttle po 5 ms, techniky počítání po jednotkách a techniky vzdálenosti po 5 px. Z experimentu vzešly následující poznatky:

- `throttle` – Čas menší než 20 ms neušetřil téměř žádná data a čas větší než 30 ms způsoboval velké skoky při rychlém pohybu myši. Dokonce i v čase 20 ms lze na obrázku 7.3 pozorovat nepřesnou rekonstrukci oproti technice vzdálenosti. Při pomalém pohybu myši jsou současně vzorkovány téměř všechny hodnoty.

[1] https://css-tricks.com/debouncing-throttling-explained-examples



Obrázek 7.3: Na obrázku jsou demonstrovány 3 techniky vzorkování dat pohybu myši s nastavenými parametry vzorkování a počtem vykreslených vzorků. Technika `nosample` nevzorkuje data a její dráha pohybu není z důvodu přehlednosti vykreslena.

- **`counting`** – Nejmenší počet, po kterém dává smysl vzorkovat data, je 2. Už při tomto počtu lze na obrázku 7.3 pozorovat nejvíce nepřesnou rekonstrukci při rychlém pohybu.
- **`distance`** – Vzdálenost větší než 10 px nepřesně rekonstruuje zatáčky při pomalém pohybu. Zároveň se zmenšující se vzdáleností výrazně klesá úspora dat.

Technika, která vyžaduje nejmenší počet dat pro nejpřesnější rekonstrukci dráhy pohybu myši, je technika vzdálenosti s nastavenou hodnotou 10 px. Současně jsem pozoroval, že při zvyšující se rychlosti pohybu myši je prohlížečem vyvoláno méně událostí. Což je důvod, proč technika počítání měla při rychlém pohybu nejhorší výsledky rekonstrukce.

7.1.4 Export informací na server

Export informací na server zabezpečuje balíček `exporter`. Tento balíček získá informace prostřednictvím balíčků `extensions`, `static` a `behavioral`, jak je navrženo v digramu balíčků 7.2. Takto získané informace následně odešle na server. Data se na server budou odesílat v rámci jedné relace pouze jednou, kdy doba relace je určená přítomností hodnoty v rámci úložiště `sessionStorage`. Mezi sekundární zodpovědnosti balíčku patří vytvoření identifikátoru uživatele, zajištění spolehlivosti běhu kódu, změření doby běhu kódu a zajištění odeslání dat i přes dočasný výpadek internetového připojení.

Vytvoření identifikátoru uživatele

Identifikátor uživatele bude vytvořen z důvodu jednoduchého párování získaných otisků a uložen bude ve standardních úložištích `localStorage`, `indexedDB` a `document.cookie`. Pokud se stane, že identifikátor bude z nějakého úložiště smazán, ale současně bude přítomen v jiném úložišti, bude nalezená hodnota rozšířena zpět do ostatních úložišť. Pokud nebude identifikátor nalezen v žádném úložišti, bude vytvořen nový a rozšířen do všech úložišť. Identifikátor bude při odeslání dat přidán k získaným informacím.

Zajištění spolehlivosti a měření doby běhu kódu

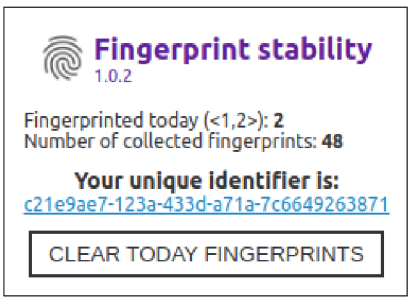
Zajištění spolehlivost běhu kódu bude realizováno pomocí konstrukce `try...catch`. Každý zdroj informací bude volán v rámci této konstrukce, a pokud dojde k nečekané výjimce, bude vrácen speciální kód indikující běhovou chybu. Současně při běhu kódu bude měřen čas jeho vykonávání pomocí funkce `performance.now`.

Kontrola stavu připojení

V případě, kdy dojde k úmyslnému odpojení od sítě nebo pouze k výpadku připojení, existuje riziko ztráty informací. Stav připojení indikuje atribut `navigator.onLine`, který však může být zfalšován. Z tohoto důvodu bude provedeno optimistické odeslání dat bez kontroly připojení, a pokud odeslání selže a hodnota tohoto atributu bude `false`, bude zaregistrován odposlech události na změnu stavu připojení. V momentě, kdy bude připojení opět dostupné, data se automaticky odešlou na server znovu.

7.2 Webové rozšíření

Účel webového rozšíření spočívá v dlouhodobém získávání informací od jednoznačně identifikovaných uživatelů, kteří si nainstalují webové rozšíření do svých prohlížečů. Jednoznačná identifikace bude zajištěna pomocí perzistentně uloženého náhodně vygenerovaného identifikátoru. Na základě dlouhodobého získávání informací bude možné zjistit, které informace se v prohlížeči běžným uživatelům mění a případně ke kolika změnám za sledované období došlo. Informace budou získávány dvakrát denně, a to dopoledne a odpoledne. Návrh uživatelského rozhraní webového rozšíření je představen na obrázku 7.4.



Obrázek 7.4: Návrh uživatelského rozhraní webového rozšíření. Uživatel má dostupné informace, kolikrát v daném dni již byly získány informace a také kolikrát byly informace již získány za celou dobu. Mimo jiné může uživatel smazat příznak, že v daném dni již byly získány informace, čímž dojde k získání dalších informací v daném dni.

7.3 Server

Server zpřístupňuje skript pro získávání informací a přijímá data zaslaná skriptem, která následně ukládá do databáze. Při vyžádání skriptu server zkontroluje, zda prohlížeč v požadavku zaslal i soubor cookie dříve nastavený serverem, a pokud soubor cookie není přítomný, vytvoří server nový a přidá cookie do odpovědi spolu se skriptem. Skript komunikuje se serverem prostřednictvím REST API, které je definováno v tabulce 7.1.

Server přijímá data ze skriptu ve formátu JSON. Server do databáze data zejména zapisuje a mezi daty nejsou modelovány žádné relace. Z těchto důvodů bude vhodné použít NoSQL databázi, která podporuje strukturovaná data.

Metoda	Endpoint	Popis
GET	/	Vráceni počtu záznamů v databázi.
GET	/scripts/tracker	Nastavení cookie a vrácení skriptu.
GET	/api/errors	Uložení běhové chyby celého skriptu.
POST	/api/nonBehavioral	Uložení statických a anomálních informací.
POST	/api/inconsistency	Spočítání a vrácení HTTP nekonzistencí.
POST	/api/keyboard	Uložení informací o interakcích s klávesnicí.
POST	/api/mouse	Uložení informací o interakcích s myší.
POST	/api/touch	Uložení informací o interakcích s dotykem.
POST	/api/extension	Uložení informace z webového rozšíření.
GET	/api/extension	Vrácení informací o webových rozšíření.
Následující endpointy vyžadují autentizaci a autorizaci uživatele.		
GET	/api/download/nonBehavioral	Vrácení statických a anomálních informací.
GET	/api/download/extension	Vrácení informací o webových rozšířeních.

Tabulka 7.1: Rozhraní REST, prostřednictvím kterého lze komunikovat se serverem.

Kapitola 8

Implementace systému

Kapitola implementace sytému vychází z návrhu systému představeném v kapitole 7 a také z dalších nastudovaných znalostí z kapitol zabývajících se získáváním informací. Popsány jsou použité technologie a přístupy k řešení jednotlivých aspektů vývoje systému včetně zdůvodnění jejich výběrů. Výsledkem této kapitoly je zejména hotová implementace knihovny na získávání informací, která má sama o sobě potenciál nahradit aktuálně nejpoužívanější knihovnu FingerprintJS.

8.1 Struktura systému

Systém na získávání informace je složen z knihovny, webového rozšíření a serveru, kde webové rozšíření a server využívají některé části kódu knihovny. Pro jednoduché a efektivní sdílení kódu mezi jednotlivými částmi bude struktura systém implementována jako monolitický repozitář (dále jen „monorepo“).

Pro správu monorepa je využit nástroj Lerna[1]. Lerna jednotlivé části systému spravuje v separátních adresářích lokalizovaných typicky v adresáři `packages`. Lerna zrychluje instalaci závislostí a optimalizuje potřebné místo na disku, protože sdílené závislosti jsou instalovány pouze jednou v kořenovém adresáři projektu. Lerna také varuje při použitím různých verzí stejných knihoven napříč částmi systému. Struktura systému spravovaného nástrojem Lerna je zobrazena na obrázku 8.1. Hlavní nevýhoda nástroje Lerna spočívá v nemožnosti synchronizovaných aktualizací[2] verzí závislostí napříč částmi systému. Řešení problému spočívá v ruční aktualizaci, nebo v použití knihovny Lerna Update Wizard[3].

```
/
├── lerna.json ..... Konfigurace monorepa.
└── packages
    ├── tracker ..... Implementace knihovny.
    ├── extension ..... Implementace webového rozšíření.
    └── backend ..... Implementace serveru.
```

Obrázek 8.1: Struktura systému ve formě monorepa spravovaného nástrojem Lerna.

[1] https://github.com/lerna/lerna
[2] https://github.com/lerna/lerna/issues/2142
[3] https://github.com/Anifacted/lerna-update-wizard

8.2 Implementační jazyk systému

Při implementaci rozsáhlejšího systému, ve kterém je kód sdílený napříč různými částmi systému, je v dlouhodobém horizontu obtížné zajistit korektní práci s objekty, které jsou dynamicky typovány. Částečné řešení může spočívat v přidání dokumentačních komentářů, které však v praxi často zastarávají a které nemusí vždy dostačovat pro popis komplexnějších objektů a vazeb v systému. Robustnější řešení představuje jazyk TypeScript[4], který staví na jazyce JavaScript a přidává k němu statickou kontrolu typů. Vytvořené datové typy v jazyce TypeScript jsou při transpilaci do jazyka JavaScript zcela vypuštěny, takže nedochází ke zvýšení velikosti kódu. Další výhoda TypeScriptu spočívá v lepší podpoře inteligentního dokončování kódu, které zvyšuje produktivitu. Z těchto důvodů je systém implementovaný v jazyce TypeScript.

Konfigurace jazyka TypeScript je realizována pomocí souboru `tsconfig.json`. Významným atributem konfigurace je `target`, který definuje verzi specifikace ECMAScript, do které bude kód transpilován. Pro zajištění podpory prohlížeče Internet Explorer 9 je nastavena verze specifikace ECMAScript 5. Knihovna využívá rozhraní `Promise` a `Fetch`, pro které TypeScript podporu u starších prohlížečů nezajistí. Podpora těchto rozhraní je zajištěna zahrnutím kódu typu polyfill[5].

8.3 Knihovna pro získávání informací

Knihovna je rozdělena na dvě základní části. Jedna část knihovny, jejíž zodpovědností je práce s informacemi, byla navržena v rámci diagramu balíčků 7.2 a je lokalizována v rámci adresáře `src`. Druhá část knihovny, lokalizovaná v adresáři `playground`, existuje z důvodu efektivního vývoje a okamžitého otestování implementované funkcionality z adresáře `src`.

- **Adresář `src`** – Kompilaci zdrojového kódu knihovny realizuje nástroj Rollup[6]. Při každé kompilaci jsou automaticky vygenerovány 4 druhy formátů[7] výsledného skriptu, aby bylo možné knihovnu jednoduše použít v řadě různých aplikací. Při kompilaci knihovny je také automaticky vygenerován soubor se všemi definovanými datovými typy a rozhraními v jazyce TypeScript.
- **Adresář `playground`** – Kód implementovaný v tomto adresáři není zahrnut do kompilace zdrojového kódu knihovny. Testování funkcionality je realizováno pomocí nástroje Webpack[8]. Webpack umožňuje importovat veškerou funkcionalitu z adresáře `src` a také spustit lokální server využívající protokol HTTPS. Server se při každé uložené změně v jakémkoliv adresáři automaticky restartuje a ihned reflektuje provedené změny.

V adresáři `playground` je implementován algoritmus, který dokáže vyhodnotit počet nekonzistencí a anomálií a jehož výsledky byly demonstrovány v tabulce 5.5. Dále je zde také implementován algoritmus, pomocí něhož byl realizován experiment pro volbu techniky vzorkování dat z pohybu myši popsaný v kapitole 7.1.3. Zbytek této kapitoly se nadále bude věnovat zejména popisu implementace zdrojů informací nacházejících se v adresáři `src`.

[4] https://www.typescriptlang.org
[5] https://github.com/taylorhakes/promise-polyfill, https://github.com/github/fetch
[6] https://rollupjs.org
[7] Vygenerovány jsou formáty `iife`, `umd`, `cjs` a `es`. Formát `iife` je dostupný skrze REST API serveru.
[8] https://webpack.js.org

8.3.1 Objekty `navigator` a `screen`

Oba objekty `navigator` (kapitola 3.1) a `screen` (kapitola 3.3) obsahují velké množství atributů, ze kterých je možné získat informace. Mnoho atributů je však značeno jako experimentálních, zastaralých nebo nestandardních. Ke všem těmto různě značeným atributům bude referováno jako k nestandardním, protože nejsou implementovány ve všech prohlížečích a protože v případě experimentálních atributů může v budoucnu dojít ke změně jejich chování. Rozdělení atributů na standardní a nestandardní je pro tyto objekty realizováno v tabulce 8.1.

Typ	**Atributy**
`navigator`	
Standardní	`userAgent`, `platform`, `appVersion`, `language`, `languages`, `hardwareConcurrency`, `plugins`, `mimeTypes`, `mediaDevices`, `appCodeName`, `appName`, `cookieEnabled`, `maxTouchPoints`, `doNotTrack`, `javaEnabled`, `onLine`, `product`, `productSub`, `vendor`, `vendorSub`
Nestandardní	`oscpu`, `cpuClass`, `userLanguage`, `browserLanguage`, `msDoNotTrack`, `msMaxTouchPoints`, `battery`, `connection`, `keyboard`, `gamepads`, `bluetooth`, `xr`, `deviceMemory`, `wakeLock`
`screen`	
Standardní	`devicePixelRatio`, `width`, `height`, `availWidth`, `availHeight`, `colorDepth`, `pixelDepth`, `orientation`
Nestandardní	`availLeft`, `availTop`

Tabulka 8.1: Rozdělení atributů v objektech `navigator` a `screen` na standardní a nestandardní.

Mezi nové techniky získávající informace o zobrazovacím zařízení patří detekce obnovovací frekvence (kapitola 3.3.1), detekce kvality zobrazovacího zařízení (kapitola 3.3.2) a detekce podpory dotyku v jazyce CSS (kapitola 3.3.3). Při implementaci detekce obnovovací frekvence jsem narazil na nestabilitu získané informace způsobenou zatížením webového prohlížeče. Zvýšení stability jsem docílil zvýšením počtu získaných informací, ze kterých je vybrána nejfrekventovanější hodnota obnovovací frekvence. Implementace detekce kvality zobrazovacího zařízení detekuje podporu barevného gamutu `srgb`, `p3` a `rec2020`.

8.3.2 Písma

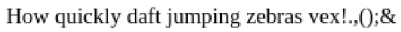
Písma jsou získána pomocí technologie ActiveX (kapitola 3.6.1), která funguje v rodině prohlížečů Trident, a pomocí jazyka JavaScript, resp. TypeScript (kapitola 3.6.2). Při implementaci detekce jazykem TypeScript jsem nastavil velikost písma na 72 px a také jsem implementoval porovnání rozměrů vůči 3 výchozím písmům. Jako text prvku `span` jsem zvolil řetězec `"mmmliMwWeLliIROO&1~<>$#"`, protože kombinuje více dříve představených vzorů a navíc obsahuje i další speciální znaky. Při vytvoření prvku `span` jsou také resetovány všechny styly, které by mohly mít vliv na velikost prvku. Celkem je detekováno 74 písem, které byly převzaty ze studií [50, 29, 17], ze skriptu FingerprintJS a 5 nových písem[9] jsem přidal ze svého operačního systému `Ubuntu 20.04`.

[9] Přidaná nová písma jsou Ubuntu, Kalapi, Gayathri, Dyuthi a Sawasdee.

8.3.3 Prvek canvas

Z prvku canvas je získáno celkem 10 různých informací. Jeden druh informace pracuje s vyhlazováním písma (kapitola 3.8.1), při kterém jsou analyzovány jednotlivé pixely, resp. sub-pixely prvku canvas. Zbylých 9 zdrojů informací je založeno na 8 izolovaně implementovaných technikách a jedné technice, která všech 8 kombinuje. Hlavní důvod tohoto rozdělení spočívá v očekávaném zvýšením stability získané informace, jak bylo uvedeno v kapitole 7.1.2. Výsledky použitých technik jsou zobrazeny na obrázku 8.2. Knihovna umožňuje získat jednotlivé informace přímo v kódování Base64, nebo na takto zakódované informace navíc aplikuje hašovací funkci.

(a) Písmo Arial.

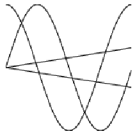


(b) Záložní písmo.

(c) Emodži.

(d) Vyplňování oblastí.

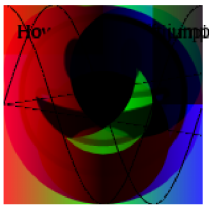
(e) Míchání barev.



(f) Matematické funkce.

(g) Lineární barevný přechod.

(h) Transformace posunu a rotace.



(i) Výsledek aplikace technik 8.2a–8.2h.

Obrázek 8.2: Demonstrace výsledků 9 technik, které slouží k získání informací. U každé techniky lze konfigurovat velikost prvku canvas, která ovlivní velikost vykresleného obsahu. Obrázky technik byly vygenerovány v adresáři `playground`.

8.3.4 Detekce architektury pomocí instrukční sady

U detekce architektury pomocí instrukční sady popsané v kapitole 3.9.1 byly provedeny následující dvě úpravy původního algoritmu, protože po dobu, co jsou oba kódy vykonávány, není možné interagovat s webovou stránkou.

- Byl snížen počet iterací na 200 000 z původních 10 000 000. Ke snížení došlo, protože výpočet v rodině prohlížečů Trident trval několik sekund, zatímco na stejném zařízení v prohlížeči Chrome trval výpočet pouze několik stovek milisekund.
- Další úprava spočívala v definici maximální doby 800 ms, po kterou je možné vykonávat jednotlivé kódy.

Vzhledem k těmto úpravám původního algoritmu nebudou výsledky použity pro detekci architektury. Výsledky doby běhu dvou kódů budou zaokrouhleny na nejbližších 200 ms a bude analyzováno, zda je možné na základě těchto výsledků v praxi odlišit uživatele.

8.3.5 Detekce snížení přesnosti časových údajů

Detekce snížení přesnosti časových údajů využívá časové údaje z rozhraní `Date.getTime`, `performance.now` a `Event.timeStamp`. Časové údaje jsou získány ze všech rozhraní celkem dvakrát, kdy druhé získání je provedeno asynchronně pro snížení pravděpodobnosti falešně pozitivního výsledku detekce. Pokud je všech 6 získaných hodnot dělitelných 100, pak je detekováno snížení přesnosti časových údajů.

8.3.6 Získávání informací z klávesnice, myši a dotyku

Získávání informací z klávesnice, myši a dotyku může uživatel knihovny realizovat dvěma způsoby. Uživatel může získat informace bezprostředně poté, co nastane interakce, nebo v rámci mikro-dávek popsaných v kapitole 7.1.3.

Implementace získávání informací z klávesnice implementuje přepínač kódování znaků, kterým lze omezit míra rizika úniku citlivých informací. Pokud je kódování znaku zapnuté, pak není předána hodnota fyzicky stisklé klávesy, ale pouze lokace klávesnice, kde došlo ke stisku klávesy.

Implementace získávání informací z myši implementuje přepínač vzorkování, kterým je možné optimalizovat množství nezbytně získaných informací. Implementace vzorkování je založena na technice vzdálenosti, která v experimentu realizovaném v kapitole 7.1.3 dosahovala nejpřesnější rekonstrukce dráhy pohybu myši.

Veškeré zdroje informací získávají souřadnice z atributů `clientX` a `clientY`, které nejsou ovlivněny pozicí okna prohlížeče ani úrovní posunu na stránce. Čas vzniku události je získán z atributu `timeStamp`. Získávána je i informace o všech současně stisknutých modifikačních klávesách, která je zakódována do jednoho čísla.

8.4 Webové rozšíření

Webové rozšíření je implementováno pro dlouhodobě nejpoužívanější[10] prohlížeč Chrome. Uživatelské rozhraní, jak bylo navrženo na obrázku 7.4, je implementováno knihovnou React[11]. Identifikátor uživatele je uložen prostřednictví rozhraní `chrome.storage.local`, ke kterému je nutné získat povolení `"storage"` specifikované v konfiguračním souboru `manifest.json`.

Webové rozšíření je publikováno[12] v internetovém obchodu Chrome, odkud ho je možné instalovat do prohlížeče. Při vytvoření účtu v internetovém obchodu Chrome bylo nutné zaplatit registrační poplatek $5. Celkem byly publikovány tři verze rozšíření, kde jejich kontroly zabraly 3–4 pracovní dny.

8.5 Server

Server a rozhraní REST je implementováno pomocí frameworku Express[13]. Express je malý, rychlý a dostatečně vyspělý framework, pro který existuje mnoho kvalitních, udržovaných rozšíření. Mezi použitá rozšíření se řadí zpracování souborů cookies, zpracování JSON těla požadavku nebo nalezení adresy IP v přijatých hlavičkách požadavku.

[10] https://www.w3counter.com/globalstats.php?year=2021&month=2

[11] https://reactjs.org

[12] https://chrome.google.com/webstore/detail/fingerprinting/lncjdmikbedgeehmcgpjkdgokagoibjl

[13] https://github.com/expressjs/express

Skript získávající informace komunikuje se serverem skrze rozhraní `Fetch` z různých domén. Součástí zaslaných informací skriptem musí být i soubor cookie nastaven serverem, pokud tyto soubory cookie prohlížeč nezakázal. Aby byl soubor cookie do požadavku zahrnut, je nutné provést explicitní konfiguraci v rozhraní `Fetch` i v hlavičkách odpovědi zaslaných serverem. V rozhraní `Fetch` došlo k zahrnutí pověření a na straně serveru byl implementován middleware, který umožňuje přijímat data napříč různými doménami spolu s pověřeními.

8.5.1 Nasazení serveru a databáze

Pro nasazení serveru a databáze jsem využil platformu Firebase[14] vyvíjenou firmou Google. Firebase nabízí mnoho komplexních služeb. Služby mají v základu určité kvóty[15] využití, které jsou zdarma, případné platby jsou účtovány za využití, nikoliv paušálně. Tato práce využívá následujících služeb:

- **Firebase Hosting** – Místo pro nasazení serveru a získání doménového jména[16].
- **Cloud Functions** – Zajištění přístupu k příchozím a odchozím HTTP požadavkům.
- **Cloud Firestore** – Škálovatelná dokumentová NoSQL databáze.

Jelikož server využívá Cloud Functions, které nejsou dostupné v rámci bezplatného plánu Spark, musel jsem přejít do plánu Blaze. Plán Blaze v sobě zahrnuje veškeré výhody plánu Spark a navíc umožňuje využívat služby i nad jejich bezplatné kvóty. Přechod na plán Blaze s sebou tedy nese i určité poplatky za využití služeb, avšak tyto poplatky se v případě této práce pohybují v řádech nižších desítek korun.

Dokumentace platformy a služeb je velmi podrobná, až na jednu výjimku, na kterou jsem narazil. Využití služeb Firebase Hosting a Cloud Functions povoluje práci pouze s jedním souborem cookie, jehož jméno musí být právě `__session`[17]. Při lokálním vývoji v emulátoru Firebase služeb jsou však propagovány všechny soubory cookie.

8.6 Zajištění kvality kódu

Kvalita kódu je na úrovní kompilace a kompatibility jednotlivých částí systému zajištěna jazykem TypeScript. Dalšího zvýšení kvality lze dosáhnout prostřednictvím nástrojů pro statickou analýzu kódu a vytvořením automatizovaných testů.

8.6.1 Statická analýza kódu

Statická analýza kódu je provedena nástrojem ESLint[18], který podporuje jazyk JavaScript i TypeScript. Konfigurace nástroje je realizována v kořenovém adresáři projektu v souboru `.eslintrc.js`. Každá část systému může implementovat vlastní soubor, ve kterém je možné přepsat nebo doplnit konfiguraci z kořenového adresáře podle specifických potřeb.

ESLint jsem do systému začlenil až v průběhu vývoje, kdy jsem pomocí něho byl schopen detekovat a opravit přes 60 varování a 10 chyb. Do nástroje ESLint jsem dále integroval následující dvě rozšíření:

[14] https://firebase.google.com
[15] https://firebase.google.com/pricing
[16] https://online-fingerprinting.web.app
[17] https://stackoverflow.com/a/44935288
[18] https://eslint.org

- **SonarJS**[19] – SonarJS dokáže detekovat pokročilé chyby a špatné vzory v kódu. Díky tomuto rozšíření jsem provedl refaktoring 5 funkcí, které měly velkou kognitivní komplexitu, tj. byly obtížné na pochopení.
- **Prettier**[20] – Prettier slouží k zajištění konzistentního formátování kódu.

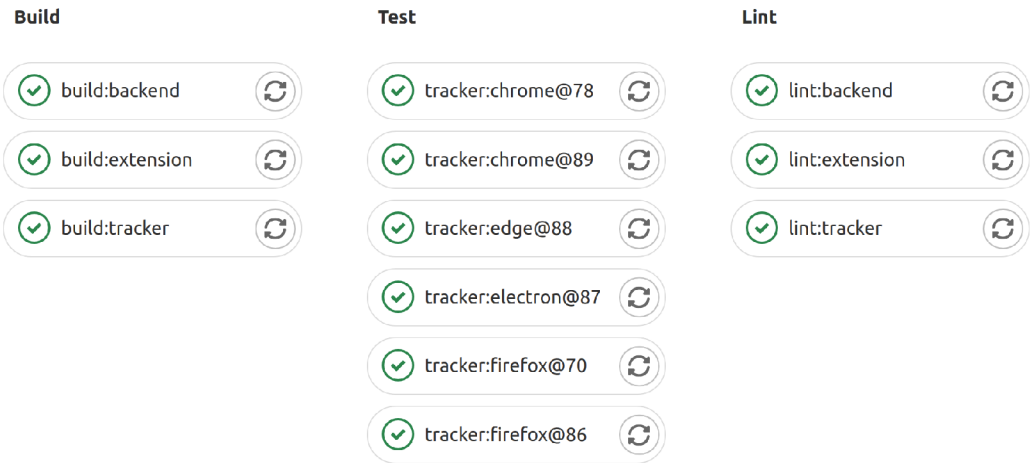
8.6.2 Testy

Testování knihovny je implementováno nástrojem Cypress[21]. Cypress aktuálně nabízí podporu testování v rodinách prohlížečů Blink a Gecko, kde prohlížeče z těchto rodin mohou běžet i v režimu headless. Celkem jsem vytvořil 142 jednotkových testů, které pokrývají 85 % kódu knihovny. Číslo pokrytí bylo získáno pomocí knihovny Istanbul[22] v prohlížeči Chrome. Při měření pokrytí v jednom prohlížeči nelze dosáhnout 100 % pokrytí kódu, protože kód knihovny obsahuje větvení specifické vždy pouze pro jednu rodinu prohlížečů.

8.6.3 GitLab CI

Proces sestavení systému, testování knihovny a statické analýzy kódu systému je automatizován prostřednictvím GitLab CI[23]. Proces je rozdělen do 3 sekvenčně prováděných etap (angl. stages), ve kterých paralelně probíhají specifikované „jednotky práce“ (angl. jobs). Konfigurace procesu je realizována v souboru `.gitlab-ci.yml` a výsledek provedení všech etap v prostředí GitLab je zobrazen na obrázku 8.3.

Provedení celého procesu trvalo v průměru 35 minut. V rámci optimalizace doby běhu jsem využil klíčového slova `needs`[24], které umožňuje provádět „jednotky práce“ mimo pořadí na základě definovaných závislostí. Tato optimalizace zrychlila provádění procesu o 32 %.



Obrázek 8.3: Úspěšné provedení etap sestavení systému, testování knihovny a statické analýzy kódu systému v GitLab CI. Knihovna je automatizovaně testována ve 4 různých prohlížečích, kde v prohlížečích Chrome a Firefox jsou použity i starší verze pro otestování kompatibility kódu.

[19] https://github.com/SonarSource/eslint-plugin-sonarjs
[20] https://github.com/prettier/eslint-plugin-prettier
[21] https://www.cypress.io
[22] https://istanbul.js.org
[23] https://docs.gitlab.com/ee/ci
[24] https://docs.gitlab.com/ee/ci/yaml/README.html#needs

Kapitola 9

Analýza získaných informací

Kapitola se věnuje analýze získaných dat prostřednictvím implementované knihovny. Na začátku kapitoly je představena vytvořená datová sada a příprava dat pro analýzu. Následně je věnována pozornost samotné analýze získaných dat z hlediska míry získané informace, rychlosti jejich získání a nakonec i stability. Analýza dat je realizována v jazyce Python.

9.1 Datová sada

Implementovaná knihovna byla od 1.3.2021 postupně nasazena na 4 komerční webové stránky, které svými produkty cílí zejména na lokální český trh. Analýza získaných informací je realizována 6.4.2021. Za více než měsíc bylo získáno celkem **7 632** otisků a nebyla zaznamenána žádná běhová chyba kódu knihovny.

9.1.1 Příprava dat

Před samotnou analýzou dat byla provedena příprava dat, která se sestávala z fází popsaných v následujícím seznamu.

1. **Výběr dat** – V rámci analýzy budou použita data pouze od unikátních uživatelů, jelikož duplicitní data stejných uživatelů by negativně ovlivnila výsledky analýzy míry získané informace. Uživatel má přiřazený identifikátor v rámci každé domény a také mezi doménami pomocí souboru cookie třetí strany. I přes odstranění známých duplicitních záznamů se mohou v datové sadě nacházet duplicitní záznamy uživatelů, kteří například pravidelně čistí paměť svého prohlížeče nebo používají anonymní režim v prohlížečích.
   (a) Odstraněno 1 787 záznamu podle identifikátoru vytvořeného u uživatele.
   (b) Odstraněno 2 200 záznamu podle souboru cookie třetí strany.
2. **Čištění dat** – Po fázi výběru dat jsou smazána data, která mají neplatnou hodnotu, nebo pochází od robotů.
   (a) Odstraněno 10 záznamů s neplatnou hodnotou
   (b) Odstraněno 282 záznamů od robotů identifikovaných řetězcem `User-Agent`.
3. **Transformace dat** – Fáze transformace dat mění strukturu získaných dat. Použita je knihovna user_agents[1] pro analýzu řetězce `User-Agent`. Transformace zahrnuje

[1] https://pypi.org/project/user-agents

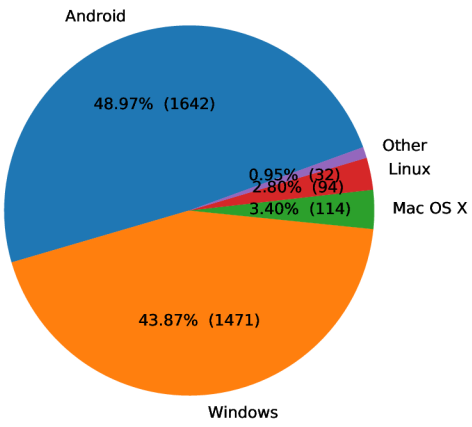
i sjednocení formátů dat, které se v rámci daného měsíce, kdy byla knihovna nasazena, změnily. V neposlední řadě je z informací o baterii detekována její přítomnost.

4. **Stabilizace dat** – Fáze stabilizace zvyšuje stabilitu dat v čase, ale současně může snížit míru získané informace. Mezi implementované stabilizace patří například ponechání pouze majoritních verzí v řetězci `User-Agent`, úprava rozměrů obrazovky na základě úrovně přiblížení nebo rotace zařízení a získání rozložení klávesnice dle mapování kláves.

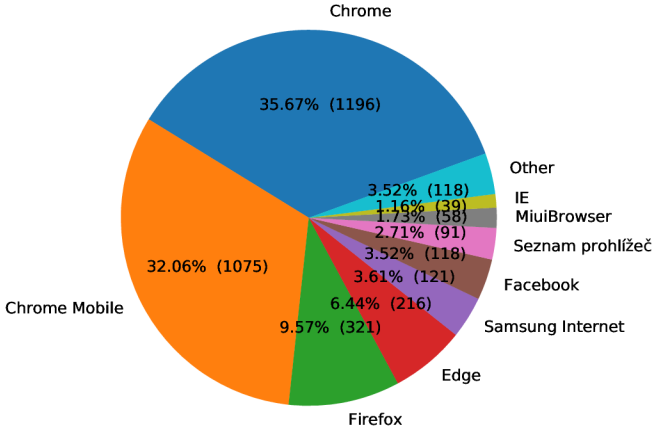
5. **Vytvoření nových informací** – Počítače, které obsahují baterii, jsou kategorizovány jako notebooky.

9.1.2 Popis datové sady

Na základě provedené přípravy dat zůstalo v datové sadě **3 353** potenciálně unikátních záznamů od uživatelů. Popis datové sady je založen na analýze řetězce `User-Agent`. Mobilních zařízení je v datové sadě 45 %, osobních počítačů 39 %, notebooků 12 % a zbylé 4 % připadají tabletům. Na obrázku 9.1 je zobrazeno zastoupení operačních systémů a na obrázku 9.2 zase zastoupení prohlížečů v datové sadě.



Obrázek 9.1: Zastoupení operačních systému v datové sadě.



Obrázek 9.2: Zastoupení webových prohlížečů v datové sadě.

9.2 Míra získaných informací

Pro vyjádření míry získané informace se používá informační entropie. Informační entropie vyjadřuje míru nejistoty s jakou pravděpodobností může daná veličina nabýt hodnoty. Aby bylo možné porovnávat výsledky informační entropie mezi různě velkými datovými sadami otisků, používá se normalizovaná informační entropie. Výpočet normalizované entropie je realizován následovně:

$$NH = \frac{H(X)}{H_M} = \frac{-\sum_{i=1}^{N} P(x_i) log_2 P(x_i)}{log_2 N} \tag{9.1}$$

kde čitatel $H(X)$ značí entropii, ve které proměnná X nabývá hodnot $\{x_1, x_2, ..., x_N\}$ a $P(X)$ značí pravděpodobnost výskytu hodnoty. Jmenovatel H_M značí maximální možnou entropii, ve které jsou pravděpodobnosti výskytů všech hodnot stejné. Proměnná N reprezentuje počet získaných otisků v datové sadě.

9.2.1 Informace s nejvyšší entropií

Normalizovaná entropie všech informací v datové sadě z této práce je rovna 0.99, což je velmi vysoké skóre. Z 3 353 otisků je 96 % otisků rozdílných (vyskytujících se alespoň jednou) a 94 % otisků unikátních (vyskytujících se právě jednou). V tabulce 9.1 jsou zobrazeny podrobnější výsledky analýzy 20 informací s nejvyšší entropií. Pro docílení přesnějších výsledků jsou informace rozděleny podle typu zařízení, ze kterého byly získány [17].

Počítač a notebook				**Mobilní zařízení a tablet**			
Informace	**Entropie**	**Norm. entropie**	**Rozdílné hodnoty**	**Informace**	**Entropie**	**Norm. entropie**	**Rozdílné hodnoty**
Písma	6.73	0.63	473	Prvek canvas	7.36	0.69	380
WebGL renderer a vendor	6.03	0.56	270	User-Agent	6.79	0.63	344
Prvek canvas	5.89	0.55	290	Síťové informace	6.35	0.59	456
Šířka a výška obrazovky	5.38	0.50	260	WebGL renderer a vendor	5.10	0.48	71
HTTP hlavičky	5.09	0.47	226	HTTP hlavičky	4.86	0.45	238
User-Agent	5.09	0.47	155	Šířka a výška obrazovky	4.59	0.43	106
Preferované jazyky	4.10	0.38	171	Konfigurace zvuku	4.00	0.37	57
Síťové informace	3.80	0.35	246	WebGL rozšíření	3.84	0.36	67
WebGL rozšíření	3.08	0.29	58	Preferované jazyky	3.62	0.34	159
Mediální zařízení	2.77	0.26	51	Poměr pixelů	3.11	0.29	37
Konfigurace zvuku	2.60	0.24	37	Mediální zařízení	2.83	0.26	28
Detekce architektury	2.58	0.24	17	Zpracování zvuku	2.35	0.22	25
Vyhlazování písma	2.48	0.23	32	Detekce architektury	1.86	0.17	12
Logické procesory	2.45	0.23	16	Operační paměť	1.71	0.16	5
Preferovaný jazyk	2.10	0.20	30	Jazyk UI prohlížeče	1.66	0.16	34
Pluginy	2.09	0.19	37	Preferovaný jazyk	1.61	0.15	43
Zpracování zvuku	2.07	0.19	30	Vyhlazování písma	1.48	0.14	17
Rozložení kláves	1.95	0.18	12	„Nesledovat“	1.45	0.14	3
Jazyk UI prohlížeče	1.92	0.18	26	Virtuální realita	1.35	0.13	7
Operační paměť	1.83	0.17	5	Časové pásmo	1.30	0.12	47

Tabulka 9.1: Dvacet informací s nejvyšší entropií rozdělených podle typu zařízení.

Detekce instalovaných písem je nejvíce diskriminativní informací na počítačích, avšak v případě mobilních zařízení je detekce písem až na 28. místě s normalizovanou entropií 0.05 a s pouze 6 rozdílnými množinami písem. Jedno z možných vysvětlení může být, že

na počítačích je poměrně jednoduché instalovat nová písma a současně mohou být písma instalována automaticky v rámci instalace nových programů viz příloha C.5.

Řetězec User-Agent je více diskriminativní na mobilních zařízeních než na počítačích. Jeden z důvodů je ten, že řetězce na mobilních zařízeních často obsahují i přesné názvy značek mobilních zařízení. V datové sadě jsou nejvíce zastoupeny na operačním systému Android mobilní zařízení od značek Samsung (444), XiaoMi (321) a Huawei (293).

9.2.2 Informace s nulovou entropií

Naproti informacím s vysokou entropií z tabulky 9.1 se v datové sadě vyskytují i informace s velmi nízkou, nebo dokonce nulovou entropií. Nulová entropie je způsobena tím, že daná informace nabývá vždy pouze jedné hodnoty. Jelikož na základě stejné hodnoty nelze odlišit různé uživatele, nemá smysl, aby takové informace knihovna získávala. Informace s nulovou entropií, o které může být knihovna zjednodušena, jsou pro různé typy zařízení zobrazeny v tabulce 9.2.

	Počítač a notebook	**Mobilní zařízení a tablet**
Společné	`appCodeName`, `cookieEnabled`, `localStorage`, `sessionStorage`, `snížení přesnosti časových údajů`	
Individuální	`onLine`, `devicePixelRatio`	`appName`, `product`, `vendorSub`, `availTop`, `availLeft`, `cpuClass`, `msDoNotTrack`, `msMaxTouchPoints`, `browserLanguage`, `systemLanguage`, `userLanguage`, `window.doNotTrack`

Tabulka 9.2: Informace s nulovou entropií rozdělené podle typu zařízení.

Za komentář stojí zejména nulová entropie detekce snížení přesnosti časových údajů, jejíž implementace je popsaná v kapitole 8.3.5. Veškeré zaznamenané výsledky skončily s výsledkem `false`, což znamená, že nebyly zaznamenány žádné falešně pozitivní výsledky, ale současně nelze vyloučit falešně negativní výsledky. Dle mých ručních testů jsem však v prohlížečích Tor Browser a Firefox[2] nezaznamenal žádné falešně negativní výsledky. Na základě těchto faktů usuzuji, že se v datové sadě nenachází žádný uživatel, který by na dané stránky přistupoval přes prohlížeč Tor Browser, nebo se vědomě chránil proti získávání informací v prohlížeči Firefox.

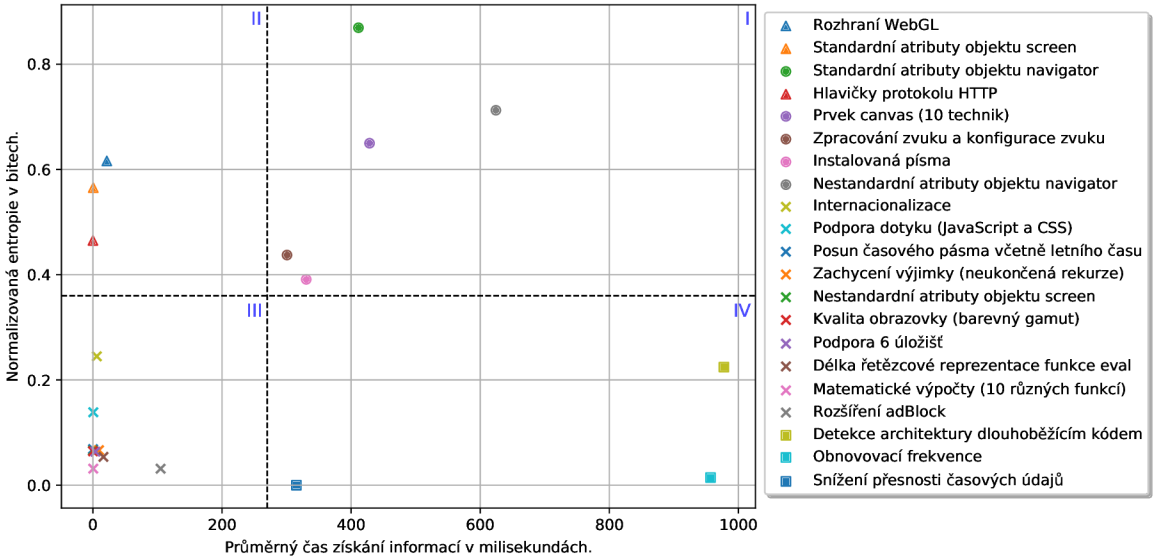
9.2.3 Normalizována entropie a rychlost získání informací

V rámci analýzy informací jsem měřil dobu, po kterou trvá získat určitou skupinu informací. Skupina vždy obsahuje tematicky podobné informace. Například jsou zde skupiny obsahující pouze standardní, nebo nestandardní informace z objektu `navigator` nebo `screen`, jejichž rozdělení je popsáno v kapitole 8.3.1. Výsledky analýzy jsou zobrazeny na obrázku 9.3.

Nejlepší poměr entropie vůči času získání dosahuje skupina informací z rozhraní WebGL. Nízká doba získání je dána tím, že k získání informací stačí pouhé přečtení atributů. Vysoká entropie je dána zase tím, že zařízení obsahují velké množství různých modelů grafických karet.

[2]S povolením možnosti `privacy.resistFingerprinting` na adrese `about:config`.

Naopak mezi nejhorší poměr entropie vůči času získání patří informace o snížení přesnosti časových údajů, která byla zhodnocena v předchozí kapitole 9.2.2, a informace o obnovovací frekvenci obrazovky. Vysoká doba získání obnovovací frekvence je zapříčiněna nutností provést několik iterací v rámci smyčky události, aby bylo získáno více odhadů obnovovacích frekvencí, ze kterých je následně vybrána ta nejčastěji naměřená hodnota. Nízká míra informace je dána tím, že 98 % obrazovek stále dosahuje obnovovací frekvence pouze 60 Hz. Druhá nejčastější obnovovací frekvence vyskytující se v datové sadě je 120 Hz. Pokud budou obrazovky s vyšší obnovovací frekvencí v budoucnu více finančně dostupné a uživatelé se rozhodnout pro jejich nákup, lze očekávat postupné zvýšení entropie.



Obrázek 9.3: Na grafu je zobrazena normalizovaná entropie vůči průměrné době získání skupiny informací. Názvy skupiny informací jsou v legendě seřazeny od nejlepšího poměru entropie vůči času získání po nejhorší. Graf jsem rozdělil do 4 kvadrantů, kde kvadranty `I` a `II` obsahují skupiny informací s vyšší entropií a kvadranty `III` a `IV` obsahují skupiny informací s nižší entropií. Skupiny informací v kvadrantech `II` a `III` lze získat velmi rychle a získání skupin informací v kvadrantech `I` a `IV` trvá delší dobu, která je způsobena vyšší výpočetní náročností.

Informace o době získání informací jsou důležité, protože čím déle trvá určitou informaci získat, tím déle dochází k blokování vykonávání kódu webové stránky, čímž může klesnout uživatelská přívětivost dané stránky. Z tohoto důvodu je dobré zabývat se optimalizací množství kódu, který je nutné vykonat, aby bylo stále možné identifikovat uživatele. Na obrázku 9.3 se ve `IV.` kvadrantu nachází 3 skupiny informací, jejichž získání trvá delší dobu a jejichž entropie je spíše nižší. Pokud by tyto skupiny knihovna nezískávala, snížila by se unikátnost otisků z 94 % na 92.7 %. Současně by však klesla průměrná doba získávání informací o 51 % na 2 496 ms. Nezískáním skupin informací ze `IV.` kvadrantu se tedy mírně sníží unikátnost otisku, ale výrazně se sníží doba získávání otisku.

9.2.4 Anomální a nekonzistentní informace

Vyhodnocení anomálních a nekonzistentních informací je realizováno jazykem TypeScript v adresáři `playground/extensions`. Data pro analýzu byla vybrána a očištěna stejně, jak

bylo uvedeno v kapitole 9.1.1. Fáze transformace, stabilizace a vytvoření nových informací nebyly realizovány z důvodu zachování maximální autenticity dat.

Po provedení prvních 2 fází přípravy dat jsou data roztříděna do 4 souborů `.json` na základě domén webových stránek. Třídění podle domén jsem realizoval, protože na jedné z webových stránek docházelo k přepsání nativních metod `Function.prototype.toString` a `Object.prototype.toString`. Tyto metody tedy byly vždy správě kategorizovány jako falešné, avšak jejich zfalšování nebylo zapříčiněno ze strany uživatele, tudíž vytvářely falešně pozitivní výsledky. Z tohoto důvodu je nutné pro budoucí nasazení knihovny v praxi zabezpečit, aby knihovna na dané webové stránce bez přítomnosti jakýchkoliv rozšíření nedetekovala žádné falešně pozitivní informace.

V první verzi nasazené knihovny byly detekovány anomální informace automatizovaně na objektech `window`, `screen` a `navigator`. Přestože některá webová rozšíření vytvářejí anomální informace přímo na objektu `window`, automatický průchod všech atributů detekoval velké množství falešných anomálií způsobených knihovnami, které daná webová stránka využívala. Z tohoto důvodu byl vytvořen seznam, který, kromě jiných funkcí, definuje 6 funkcí[3] na objektu `window`, které jsou potenciálními kandidáty na zfalšování webovými rozšířeními. V dalších verzích nasazené knihovny již jsou detekovány anomální informace automatizovaně pouze na objektech `screen` a `navigator`.

Po sloučení otisků ze všech domén bylo na základě technik popsaných v kapitole 5 a principu počítání anomálií a nekonzistencí uvedeného u tabulky 5.5 zjištěno následující:

- 6 % otisků obsahuje alespoň 1 anomální nebo nekonzistentní informaci,
- 3 % otisků obsahují alespoň 2 anomální nebo nekonzistentní informace.

Nejvíce anomálních informací bylo detekováno technikou smazání atributu z objektu (kapitola 5.1.4) a nejvíce nekonzistentních informací bylo detekováno na úrovni rodiny prohlížečů (kapitola 5.2.1).

9.3 Zjištění na základě analýzy informací

V této části kapitoly jsou představena zjištění, která jsem učinil v rámci analýzy datové sady. Mezi jednodušší, přesto zajímavá zjištění patří:

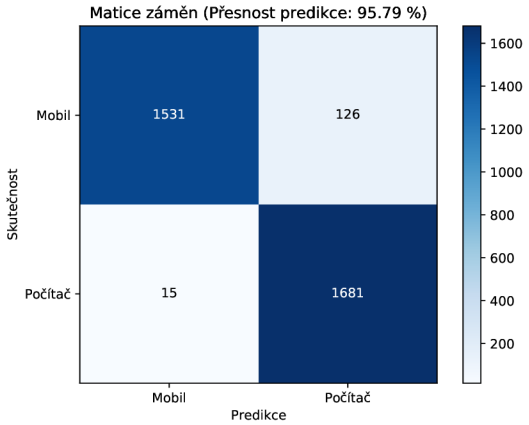
- Z výsledků matematických funkcí nelze v praxi detekovat operační systém, protože 95.85 % matematických výsledků je stejných mezi všemi operačními systémy. Tomuto odpovídá normalizovaná entropie matematických výpočtů, která je na počítačích 0.05 bitů a na mobilních zařízeních pouze 0.01 bitů.
- V rodině prohlížečů Trident je hodnota atributu `message` na události vyvolané výjimky lokalizovaná. Kromě anglického popisu nastalé situace `"Out of stack space"` je v datové sadě přítomen i český překlad `"Nedostatek místa v zásobníku"`.
- V 16.5 % prohlížečů Seznam.cz bylo přítomných více než 7 popisů připojených mediálních zařízení, které lze získat z atributu `mediaDevices`. Například následující popis `"Stolní mikrofon (Microsoft® LifeCam HD-3000)"` obsahuje přesný typ připojeného komunikačního příslušenství. Normalizovaná entropie informací o mediálních zařízeních dosahuje na počítačích i mobilních zařízeních hodnoty 0.26 bitů.

[3] `performance.now`, `Date`, `Date.now`, `Date.getTime`, `requestAnimationFrame`, `XMLHttpRequest`

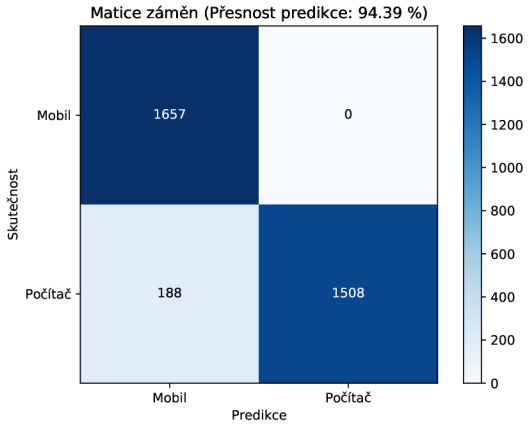
9.3.1 Predikce mobilního zařízení nebo tabletu

Mobilní zařízení a tablet je možné rozpoznat podle řetězce `User-Agent`. Pouze na zfalšování tohoto řetězce se však specializují 4 webová rozšíření, která mají tento řetězec ve svém názvu a jejichž analýza byla provedena v tabulce 5.5. Z tohoto důvodu jsem navrhl algoritmus, který s 95.79 % přesností umožňuje predikovat mobilní zařízení nebo tablet.

Algoritmus pracuje s rozměry zobrazovacího zařízení, dostupností pluginů a přítomností baterie na zařízení. Rozdíl atributů `avail{Width|Height}` od atributů `width|height` je na mobilních zařízeních nulový. Současně jsou hodnoty atributů `avail{Left|Top}` taktéž nulové. Mobilní zařízení dále neobsahují žádné pluginy získatelné pomocí iterace, nebo obsahují pouze jeden plugin s názvem `"Shockwave Flash"`. Přítomnost baterie bylo možné v 92 % případů detekovat. Podrobnější výsledky navrženého algoritmu na datové sadě jsou zobrazeny ve dvou maticích záměn na obrázku 9.4.



(a) S přesností 95.79 % je možné v datové sadě predikovat mobilní zařízení nebo tablet pomocí všech uvedených zdrojů informací.



(b) V tomto případě nebyla použita informace o přítomnosti baterie. Přesnost detekce se mírně snížila, ale zcela byly eliminovány falešně pozitivní výsledky, které baterie způsobovala.

Obrázek 9.4: Výsledky predikce mobilního zařízení nebo tabletu v datové sadě dle navrženého algoritmu.

9.3.2 Detekce rodiny prohlížečů

V kapitole 5.2.1 byly popsány 4 způsoby detekce rodiny prohlížečů. Kromě těchto způsobů lze rodina prohlížečů dále detekovat na základě techniky zpracování zvukového signálu popsané v kapitole 3.7.1. Na základě výsledků z datové sady jsem vytvořil tabulku 9.3, která mapuje zaokrouhlené výsledky zpracování zvukového signálu na rodinu prohlížečů.

9.4 Stabilita získaných informací

Stabilita informací bude vyhodnocena z informací získaných z komerčních webových stránek a současně i z dat pocházejících z webového rozšíření. Webové rozšíření[4] bylo v období 8.3.2021–11.4.2021 instalováno u 17 uživatelů a získáno bylo celkem 417 záznamů. V této

[4] https://chrome.google.com/webstore/detail/fingerprinting/lncjdmikbedgeehmcgpjkdgokagoibjl

Rodina prohlížečů (celkem prohlížečů)	**Blink** (2 936)	**Gecko** (329)	**Trident** (39)	**WebKit** (23)
Zpracování zvukového signálu	124	36	`undefined`	35
Operační systémy	Android, Windows, Mac OS, Linux	Windows, Android, Linux, Mac OS	Windows	iOS, Mac OS

Tabulka 9.3: Rodina prohlížečů lze detekovat na základě zaokrouhlených výsledků zpracování zvukového signálu. U každé rodiny prohlížečů se nachází výčet operačních systémů, na kterých byla daná rodina prohlížečů zaznamenána. Z výsledků je patrné, že zpracování zvukového signálu nezávisí na operačním systému, tedy ani na typu zařízení.

práci bude na stabilitu informace nahlíženo jako na počet různých hodnot zaznamenaných u jednoho uživatele pro danou informaci nebo příslušnou skupinu informací. Čím méně změn je u informace nebo skupiny informací zaznamenáno, tím stabilnější informace nebo skupina informací je a naopak.

Webové rozšíření bylo navrženo a implementováno z důvodu dlouhodobější analýzy stability informací, protože uložení identifikátoru na straně uživatele nemá zpravidla dlouhotrvající charakter. Přestože je identifikátor na straně uživatele uložen ve 3 různých úložištích (kapitola 7.1.4), průměrná doba jeho uložení se pro mobilní zařízení a počítače pohybovala v průměru 9 dní. Průměrná doba uložení identifikátoru v rámci webového rozšíření je rovna 21 dní, kdy tato doba je shora omezená pouze dobou dostupnosti webového rozšíření.

9.4.1 Stabilita skupin informací

Na grafu 9.3 byla zkoumána normalizovaná entropie vůči rychlosti získání skupin informací. Skupiny informací byly v rámci tohoto grafu rozděleny do kvadrantů. Z hlediska stability téže skupin informací demonstruje tabulka 9.4, kolik změn bylo u dané skupiny informací zaznamenáno. Počet změn u každé skupiny informací byl získán z obou zdrojů zvlášť v následujících krocích:

1. Data z webových stránek a webového rozšíření jsou rozdělena do skupin podle identifikátorů uživatelů.
2. Na takto rozdělené skupiny je aplikována příprava dat z kapitoly 9.1.1 vyjma fáze výběru dat. Na připravená data je následně aplikována operace `flatten`[5].
3. V rámci skupin je pro každou informaci vytvořená množina všech různých hodnot, kterých každá informace za danou dobu nabyla.
4. Ze všech skupin jsou následně vytvořeny statistiky počtu změn pro každou informaci zvlášť a pro skupinu informací.

Z výsledků v tabulce 9.4 lze pozorovat, že informace s vyšší entropií jsou náchylnější na častější změny v čase. Naopak informace s nižší entropií podléhají méně změnám v čase, a tudíž je jejich stabilita vyšší. Z tabulek 9.1, 9.4 a grafu 9.3 lze vyvodit následující závěry:

[5]Operaci `flatten` na slovníku dat realizuje knihovna https://pypi.org/project/flatten-dict.

Skupina informací	Počet změn			Kvadrant I nebo II
	Rozšíření	Web. stránky		
	Počítač	Počítač	Mobil	
Nestandardní atributy objektu navigator	261	215	318	✓
Prvek canvas (10 technik)	44	102	110	
Standardní atributy objektu screen	58	157	14	
Standardní atributy objektu navigator	76	96	48	
Detekce architektury dlouhoběžícím kódem	72	92	36	✗
Nestandardní atributy objektu screen	41	45	0	
Zpracování zvuku a konfigurace zvuku	9	26	21	✓
Rozšíření adBlock	34	8	0	✗
Hlavičky protokolu HTTP	12	28	6	✓
Rozhraní WebGL	8	24	0	
Instalovaná písma	21	4	0	
Podpora 6 úložišť	4	6	12	✗
Kvalita obrazovky (barevný gamut)	2	20	0	
Obnovovací frekvence	9	6	6	
Internacionalizace	0	0	4	
Podpora dotyku (JavaScript a CSS)	0			
Posun časového pásma včetně letního času				
Zachycení výjimky (neukončená rekurze)				
Délka řetězcové reprezentace funkce eval				
Matematické výpočty (10 různých funkcí)				
Snížení přesnosti časových údajů				

Tabulka 9.4: Tabulka demonstruje celkový počet změn u jednotlivých skupin informací rozdělených podle typu zdroje a zařízení, ze kterého informace pochází. Skupiny informací jsou seřazeny podle počtu změn. U každé skupiny informací se nachází příznak, zda skupina patří do kvadrantu `I` nebo `II`. Tyto dva kvadranty obsahují informace s vyšší entropií.

- Informace z rozhraní WebGL lze získat rychle a zároveň jsou na předních příčkách entropie u počítačů i mobilních zařízeních. Současně je počet změn na mobilních zařízeních nulový a na počítačích nedochází ke změnám příliš často. Informace z rozhraní WebGL lze tedy považovat za kvalitní zdroj informací, který například knihovna FingerprintJS neimplementuje.
- Na mobilních zařízeních představuje velmi kvalitní zdroj informací obrazovka mobilu právě díky rychlému získání, vyšší entropii a velmi nízkému počtu změn.
- Instalovaná písma na počítačích dosahují nejvyšší entropie. Vzhledem k nižšímu počtu změn představují taktéž kvalitní zdroj informací.

9.4.2 Nejvíce nestabilní informace

V této části jsou blíže popsány 4 nejvíce nestabilní skupiny informací z tabulky 9.4, které současně dosahují vyšší entropie.

- **Nestandardní atributy objektu navigator** – Nejvíce nestabilními informacemi z této skupiny jsou atributy `rtt` a `downlink` z rozhraní `connection`. Na počítačích

připadá 69 % všech změn právě těmto dvěma atributům. Na mobilních zařízeních je to až 84 %. Navazující práce by se mohla zabývat například otázkou, zda by bylo možné tyto nestabilní informace o síťovém připojení použít ke krátkodobému identifikování uživatelů podobně, jako tomu bylo v případě baterie [41].

- **Prvek canvas** – Nejvíce změn nastalo v nejkomplexnější technice z obrázku 8.2i, která kombinuje více technik. Z dat webového rozšíření vyplývá, že k nejméně změnám došlo u techniky vykreslující emodži a písmo Arial. Z dat webových stránek na počítačích došlo k 10–12 změnám u většiny technik u 3–4 uživatelů. Na mobilním zařízení došlo k nejméně změnám u techniky vyplňování oblastí, matematických výpočtů a aplikace transformace na objekty.
- **Standardní atributy objektu screen** – Změny rozměrů obrazovky odpovídají situacím, kdy uživatelé vlastní více obrazovek, na kterých prohlíží webové stránky.
- **Standardní atributy objektu navigator** – Z dat webových stránek dochází k nejvíce změnám na počítači i mobilním zařízení ve verzi prohlížeče v řetězci `User-Agent`. Z dat webového rozšíření je nejvíce změn způsobených z atributu `mediaDevices`, který indikuje připojené mediální zařízení k počítači.

Kapitola 10

Závěr

Cílem práce bylo vytvořit knihovnu pro získávání informací na webových stránkách. Vytvořená knihovna je pokryta automatizovanými testy napříč dvěma rodinami prohlížečů a obsahuje 127 informací organizovaných ve 21 skupinách. Nad rámec zadání bylo vytvořeno webové rozšíření, server a databáze pro nasazení v praxi a následné analýzy dat.

Knihovna obsahuje pokročilé techniky detekce vedlejších efektů způsobených webovými rozšířeními, jejichž účel je maskovat identitu uživatele a tím zvýšit jeho anonymitu. Funkčnost implementovaných technik byla prokázána detekcí 80 anomálních a 37 nekonzistentních informací v celkem 19 testovaných webových rozšířeních. V aktuální době však, dle mých znalostí, neexistuje knihovna, která by tyto techniky implementovala. Z tohoto důvodu je prozatím použití takových webových rozšíření pro uživatele stále přínosné.

Knihovna byla nasazena na 4 komerčních webových stránkách a za více než jeden měsíc získala informace od 3 353 potenciálně unikátních uživatelů. Z těchto uživatelů mělo až 94 % zcela unikátní otisk. Z analýzy se potvrdilo, že informace, které dosahují vysoké entropie na počítačích, nemusí dosahovat vysoké entropie na mobilních zařízeních. Další pozorování naznačuje, že informace s vyšší entropií mají tendenci k častějším změnám než informace s nižší entropií. Velmi kvalitní informace z hlediska vyšší entropie, nízké doby získání a nižšího počtu změn lze získat z rozhraní WebGL. Z výsledků analýzy také vyplývají optimalizace v podobě odebrání informací s důsledkem mírné nebo žádné ztráty unikátnosti otisku a se ziskem až dvojnásobného zrychlení získání otisku.

V průběhu práce také bylo navrženo několik technik, které nebyly dříve publikovány. Mezi významnější nově navržené techniky patří detekce kvality zobrazovacího zařízení, detekce podpory dotyku v jazyce CSS, detekce falešné nativní funkce nebo predikce mobilního zařízení a tabletu. Zároveň také byly navrženy optimalizace technik získávání informací z prvku canvas nebo podpory úložišť.

Na práci je možné navázat třemi dalšími pracemi. Jedna práce by se zabývala omezením/odstraněním vedlejších efektů u webových rozšíření, které tato práce dokáže detekovat. Další práce by mohla využít část knihovny, která získává informace o chování uživatelů a vyhodnotit použitelnosti těchto informací v praxi z hlediska míry získané informace a stability. Poslední práce by mohla na základě výsledků analýz entropie a stability vytvořit algoritmus, který dokáže rozpoznat přirozené změny ve vývoji informací a identifikovat tak uživatele po delší časové období.

Literatura

[1] *System state and capabilities*. Steering Group, listopad 2020. [cit. 8.12.2020]. Dostupné z: https://html.spec.whatwg.org/multipage/system-state.html.

[2] *Web storage*. Steering Group, listopad 2020. [cit. 9.12.2020]. Dostupné z: https://html.spec.whatwg.org/multipage/webstorage.html.

[3] Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A. et al. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2014, s. 674–689. CCS '14. DOI: 10.1145/2660267.2660347. ISBN 9781450329576.

[4] Adenot, P. a Choi, H. *Web Audio API*. W3C®, červen 2020. [cit. 20.12.2020]. Dostupné z: https://www.w3.org/TR/webaudio.

[5] Ashouri, M. A Large-Scale Analysis of Browser Fingerprinting via Chrome Instrumentation. Červenec 2019. DOI: 10.13140/RG.2.2.36700.85124/1.

[6] Balter, L. a Gibson, R. *ECMAScript® 2021 Internationalization API Specification*. Ecma International, listopad 2020. [cit. 7.11.2020]. Dostupné z: https://tc39.es/ecma402.

[7] Bellamy Royds, A., Brinza, B., Lilley, C., Schulze, D., Storey, D. et al. *Chapter 13: Painting: Filling, Stroking and Marker Symbols*. W3C®, říjen 2018. [cit. 30.1.2021]. Dostupné z: https://www.w3.org/TR/SVG/painting.html.

[8] Cabanier, R. *Blending features in Canvas*. Leden 2013. [cit. 2.2.2021]. Dostupné z: https://web.archive.org/web/20150831170558/http://blogs.adobe.com/webplatform/2013/01/28/blending-features-in-canvas.

[9] Cao, Y., Li, S. a Wijmans, E. (Cross-)Browser Fingerprinting via OS and Hardware Level Features. In:. San Diego, CA, USA: [b.n.], 2017. NDSS '17. DOI: 10.14722/ndss.2017.23152. ISBN 1-1891562-46-0.

[10] Eckersley, P. How Unique is Your Web Browser? In: *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*. Berlin, Heidelberg: Springer-Verlag, 2010, s. 1–18. PETS'10. ISBN 3642145264.

[11] Englehardt, S. a Narayanan, A. Online Tracking: A 1-Million-Site Measurement and Analysis. In:. New York, NY, USA: Association for Computing Machinery, 2016, s. 1388–1401. CCS '16. DOI: 10.1145/2976749.2978313. ISBN 978-1-4503-4139-4.

[12] Fang, L., Au, O. C. a Cheung, N. Subpixel rendering: from font rendering to image subsampling [Applications Corner]. *IEEE Signal Processing Magazine*. 2013, sv. 30, č. 3, s. 177–189. DOI: 10.1109/MSP.2013.2241311.

[13] Faulkner, S., Eicholz, A., Leithead, T., Danilo, A. a Moon, S. *HTML 5.2*. W3C®, prosinec 2017. [cit. 6.11.2020]. Dostupné z: https://www.w3.org/TR/html52/webappapis.html.

[14] Fielding, R. T. a Reschke, J. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content* [RFC 7231]. RFC Editor, červen 2014. DOI: 10.17487/RFC7231.

[15] Freiberger, A. F. Detecting User's Handedness on Web Apps, a Quick Experiment. *Medium.com*. Září 2018. [cit. 9.3.2021]. Dostupné z: https://medium.com/@ArthurFinkler/detecting-users-handedness-on-web-apps-a-quick-experiment-a71ca0481ff.

[16] Frisbie, M. *Professional Javascript® for Web Developers*. John Wiley & Sons, Inc, 2020. ISBN 978-1-119-36644-7.

[17] Gómez Boix, A., Laperdrix, P. a Baudry, B. Hiding in the Crowd: An Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. In: *Proceedings of the 2018 World Wide Web Conference*. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018, s. 309–318. WWW '18. DOI: 10.1145/3178876.3186097. ISBN 978-1-4503-5639-8.

[18] Gómez Boix, A., Frey, D., Yerom, B. a Baudry, B. A Collaborative Strategy for Mitigating Tracking through Browser Fingerprinting. In:. Listopad 2019, s. 67–78. DOI: 10.1145/3338468.3356828. ISBN 978-1-4503-6828-5.

[19] Harband, J., Guo, S. yu, Ficarra, M. a Gibbons, K. *ECMAScript® 2021 Language Specification*. Ecma International, listopad 2020. [cit. 25.10.2020]. Dostupné z: https://tc39.es/ecma262.

[20] Hauer, P. Package by Feature. Duben 2020. [cit. 11.3.2021]. Dostupné z: https://phauer.com/2020/package-by-feature.

[21] Hawryluk, Z. *How to Detect Font-Smoothing Using JavaScript*. Listopad 2009. [cit. 15.1.2021]. Dostupné z: http://www.useragentman.com/blog/2009/11/29/how-to-detect-font-smoothing-using-javascript.

[22] Ishida, R. Language tags in HTML and XML. *W3C*. Leden 2016. [cit. 9.11.2020]. Dostupné z: https://www.w3.org/International/articles/language-tags.

[23] Ishida, R. Setting language preferences in a browser. *W3C*. Srpen 2019. [cit. 10.11.2020]. Dostupné z: https://www.w3.org/International/questions/qa-lang-priorities.

[24] Jackson, D., Rivoal, F. a Jr., T. A. *Media Queries Level 5*. W3C®, červenec 2020. [cit. 4.11.2020]. Dostupné z: https://www.w3.org/TR/mediaqueries-5.

[25] Jones, F., Nichols, M. a Pappas, S. Color and Appearance. In: *Organic Coatings*. John Wiley & Sons, Inc, Srpen 2017, s. 267–292. DOI: 10.1002/9781119337201.ch19. ISBN 9781119026891.

[26] Jorgensen, Z. a Yu, T. On Mouse Dynamics as a Behavioral Biometric for Authentication. In:. New York, NY, USA: Association for Computing Machinery, 2011, s. 476–482. ASIACCS '11. DOI: 10.1145/1966913.1966983. ISBN 9781450305648.

[27] Kacmarcik, G. Keyboard Map. *W3C*. The Web Platform Incubator Community Group. Duben 2019. [cit. 15.10.2020]. Dostupné z: https://wicg.github.io/keyboard-map.

[28] Kaur, H., Zavarsky, P. a Jaafar, F. Unauthorised Data Leakage from an Organisation through Web Browser Fingerprinting Vulnerability. In:. University of Cambridge, UK: [b.n.], Prosinec 2017. DOI: 10.2053/WorldCIS.2017.0013. ISBN 978-1-908320-81-0.

[29] Kobusińska., A., Brzeziński., J. a Pawulczuk., K. Device Fingerprinting: Analysis of Chosen Fingerprinting Methods. In: INSTICC. *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS,*. SciTePress, 2017, s. 167–177. DOI: 10.5220/0006375701670177. ISBN 978-989-758-245-5.

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

[31] Laperdrix, P., Bielova, N., Baudry, B. a Avoine, G. Browser Fingerprinting: A Survey. New York, NY, USA: Association for Computing Machinery. duben 2020, sv. 14, č. 2. DOI: 10.1145/3386040. ISSN 1559-1131.

[32] Lauke, P. H. a Zolghadr, N. *Pointer Events – Level 3*. W3C®, říjen 2020. [cit. 5.11.2020]. Dostupné z: https://www.w3.org/TR/pointerevents3.

[33] Levy, E. Batch, Stream, and Micro-batch Processing: A Cheat Sheet. Leden 2021. [cit. 11.3.2021]. Dostupné z: https://www.upsolver.com/blog/batch-stream-a-cheat-sheet.

[34] Li, S. a Cao, Y. Who Touched My Browser Fingerprint? A Large-Scale Measurement Study and Classification of Fingerprint Dynamics. In: *Proceedings of the ACM Internet Measurement Conference*. New York, NY, USA: Association for Computing Machinery, 2020, s. 370–385. IMC '20. DOI: 10.1145/3419394.3423614. ISBN 978-1-4503-8138-3.

[35] Michálek, M. *CSS pixel*. Září 2014. [cit. 3.11.2020]. Dostupné z: https://www.vzhurudolu.cz/prirucka/css-pixel.

[36] Mishra, P. *Browser Engines: The Crux Of Cross Browser Compatibility*. Srpen 2019. [cit. 14.11.2020]. Dostupné z: https://www.lambdatest.com/blog/browser-engines-the-crux-of-cross-browser-compatibility.

[37] Mowery, K. a Shacham, H. Pixel Perfect: Fingerprinting Canvas in HTML5. In: Fredrikson, M., ed. *Proceedings of W2SP 2012*. Květen 2012.

[38] Nakibly, G., Shelef, G. a Yudilevich, S. Hardware Fingerprinting Using HTML5. *CoRR*. 2015, abs/1503.01408.

[39] Nielsen, H., Fielding, R. T. a Berners Lee, T. *Hypertext Transfer Protocol – HTTP/1.0* [RFC 1945]. RFC Editor, květen 1996. DOI: 10.17487/RFC1945.

[40] Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F. et al. Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting. In:. Květen 2013, s. 541–555. DOI: 10.1109/SP.2013.43. ISBN 978-1-4673-6166-8.

[41] Olejnik, L., Acar, G., Castelluccia, C. a Díaz, C. The leaking battery: A privacy analysis of the HTML5 Battery Status API. *IACR Cryptology ePrint Archive*. 2015, sv. 2015, s. 616.

[42] Pentel, A. High precision handedness detection based on short input keystroke dynamics. In: *2017 8th International Conference on Information, Intelligence, Systems Applications (IISA)*. 2017, s. 1–5. DOI: 10.1109/IISA.2017.8316380.

[43] Pentel, A. Predicting Age and Gender by Keystroke Dynamics and Mouse Patterns. In:. New York, NY, USA: Association for Computing Machinery, 2017, s. 381–385. UMAP '17. DOI: 10.1145/3099023.3099105. ISBN 9781450350679.

[44] Perry, M. *Math routines are OS fingerprintable*. Září 2014. [cit. 5.2.2021]. Dostupné z: https://gitlab.torproject.org/legacy/trac/-/issues/13018.

[45] Perry, M., Clark, E., Murdoch, S. a Koppen, G. *The Design and Implementation of the Tor Browser*. červen 2018. [cit. 15.11.2020]. Dostupné z: https://2019.www.torproject.org/projects/torbrowser/design.

[46] Phillips, A. a Davis, M. *Tags for Identifying Languages* [RFC 5646]. RFC Editor, září 2009. DOI: 10.17487/RFC5646.

[47] Pieters, S. *CSSOM View Module*. W3C®, březen 2016. [cit. 3.11.2020]. Dostupné z: https://www.w3.org/TR/cssom-view-1.

[48] Rivoal., F. *Media Queries*. W3C®, červenec 2012. [cit. 4.11.2020]. Dostupné z: https://www.w3.org/TR/css3-mediaqueries.

[49] Rivoal, F. a Jr., T. A. *Media Queries Level 4*. W3C®, červenec 2020. [cit. 4.11.2020]. Dostupné z: https://www.w3.org/TR/mediaqueries-4.

[50] Saito, T., Takahashi, K., Yasuda, K., Ishikawa, T., Takasu, K. et al. OS and Application Identification by Installed Fonts. In: *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*. 2016, s. 684–689. DOI: 10.1109/AINA.2016.55.

[51] Schepers, D., Moon, S., Brubeck, M., Barstow, A., Byers, R. et al. *Touch Events – Level 2*. Touch Events Community Group, září 2019. [cit. 4.11.2020]. Dostupné z: https://w3c.github.io/touch-events.

[52] Schwarz, M., Lackner, F. a Gruss, D. JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits. In: *NDSS*. únor 2019. ISBN 1-891562-55-X.

[53] Sheedy, J., Tai, Y.-C., Subbaram, M., Gowrisankaran, S. a Hayes, J. ClearType sub-pixel text rendering: Preference, legibility and reading performance. *Displays*. 2008, sv. 29, č. 2, s. 138–151. DOI: 10.1016/j.displa.2007.09.016. ISSN 0141-9382.

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

[55] Weiss, Y. Client Hints Infrastructure. *W3C*. Web Platform Incubator Community Group. Listopad 2020. [cit. 13.11.2020]. Dostupné z: https://wicg.github.io/client-hints-infrastructure.

[56] West, M. a Weiss, Y. User-Agent Client Hints. *W3C*. Web Platform Incubator Community Group. Listopad 2020. [cit. 13.11.2020]. Dostupné z: https://wicg.github.io/ua-client-hints.

[57] Zheng, N., Paloski, A. a Wang, H. An Efficient User Verification System via Mouse Movements. In:. New York, NY, USA: Association for Computing Machinery, 2011, s. 139–150. CCS '11. DOI: 10.1145/2046707.2046725. ISBN 9781450309486.

Příloha A

Obsah přiloženého DVD

- `/`
  - `thesis` ..... Text diplomové práce včetně zdrojových souborů.
  - `thesis-code` ..... Zdrojové kódy použité k vytvoření některých grafik v práci.
  - `fingerprinting` ..... Systém pro získávání informací v jazyce TypeScript.
    - `packages` ..... Balíčky spravované nástrojem Lerna.
      - `tracker` ..... Zdrojové kódy knihovny.
      - `extension` ..... Zdrojové kódy webového rozšíření.
      - `backend` ..... Zdrojové kódy serverové aplikace.
  - `fingerprinting-analysis` ..... Analýza získaných informací v jazyce Python.
  - `excel-article` ..... Materiály k článku z konference Excel@FIT.

Příloha B

Porovnání entropie informací s dalšími studiemi

Laperdrix et al. [31] provedl porovnání entropie informací z předešlých dvou studií [17, 30], kterých byl součástí. Obě studie získávaly pouze 17 různých informací, avšak následná analýza entropie byla provedena na značně vetším vzorku uživatelů. V tabulce B.1 jsem realizoval porovnání entropie oněch 17 informací, které tato práce taktéž získávala.

Informace	**Tato práce** (2021)		**Hiding** (2018)		**AmIUnique** (2016)	
	Entropie	Norm. entropie	Entropie	Norm. entropie	Entropie	Norm. entropie
User-Agent	8.765	0.748	7.150	0.341	9.779	0.580
Accept	0.024	0.002	0.729	0.035	1.383	0.082
Accept-Encoding	0.461	0.039	0.382	0.018	1.534	0.091
Accept-Language	3.975	0.339	2.716	0.129	5.918	0.351
Seznam pluginů	1.603	0.137	9.485	0.452	1.060	0.656
Povolení cookies	0.000	0.000	0.000	0.000	0.253	0.015
Úložiště `local` a `session`	0.037	0.003	0.043	0.002	0.405	0.024
Časové pásmo	0.761	0.065	0.164	0.008	3.338	0.198
Rozlišení obrazovky a barevná hloubka	5.283	0.451	4.847	0.231	4.889	0.290
Seznam písem	4.538	0.387	6.904	0.329	8.379	0.497
Hlavičky HTTP	2.784	0.238	1.783	0.085	4.198	0.249
Platforma	1.952	0.167	1.200	0.057	2.310	0.137
„Nesledovat“	1.420	0.121	1.919	0.091	0.944	0.056
Canvas	7.589	0.648	8.546	0.407	8.278	0.491
WebGL vendor	2.219	0.189	2.282	0.109	2.141	0.127
WebGL renderer	6.436	0.550	5.541	0.264	3.406	0.202
AdBlock	0.367	0.031	0.045	0.002	0.995	0.059
Počet otisků	3 353		2 067 942		118 934	
Unikátních otisků	85.7 %		33.6 %		88.5 %	

Tabulka B.1: Porovnání entropie, normalizované entropie a unikátnosti 17 informací.

Příloha C

Ovlivnění získávaných informací

Tato kapitola popisuje, jakým způsobem mohou být ovlivněny získávané informace. Nejčastější způsob ovlivnění je konfigurace na úrovni prohlížeče nebo operačního systému. Speciální případ ovlivnění informací představuje úroveň přiblížení na stránce nebo instalace programů do operačního systému.

Kapitola neobsahuje výčet informací, které mohou být ovlivněny aktualizací prohlížeče nebo operačního systému. Popsány nejsou ani informace ovlivnitelné výměnou hardwaru zařízení.

Informace z této kapitoly jsou důležité zejména pro rozhodnutí, zda dva nebo více rozdílných otisků může pocházet od stejného uživatele. Pokud se otisky liší v částech, u kterých existuje možnost jejich ovlivnění, lze s určitou pravděpodobností rozhodnout, jestli otisky pochází od stejného uživatele či nikoliv.

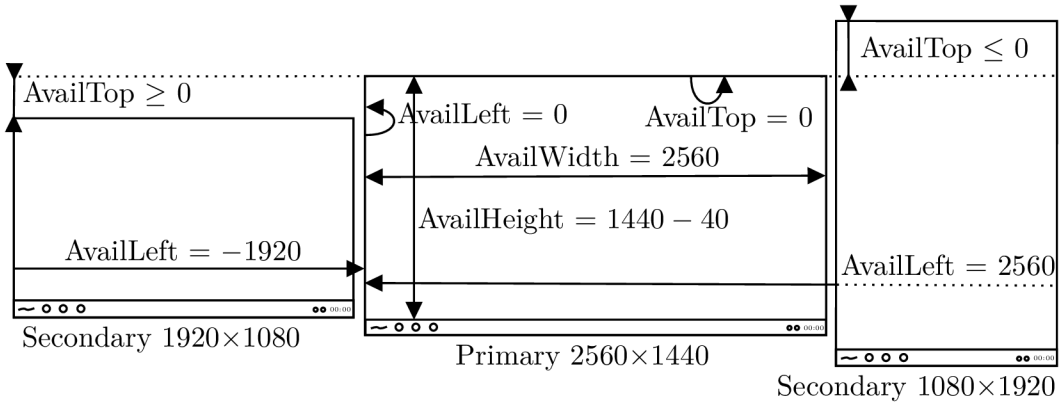
C.1 Preferované jazyky

Uživatelé prohlížečů Chrome, Edge, Firefox mohou změnit nastavení preferovaných jazyků přímo v nastavení prohlížeče. Toto nastavení nijak neovlivní nastavení preferovaných jazyků v operačním systému. Prohlížeč Safari však využívá preferované jazyky přímo ze systému, nikoliv z prohlížeče. Počáteční nastavení preferovaných jazyků v prohlížeči je realizováno při instalaci prohlížeče na základě jazyka instalačního souboru, který je často definován jazykem operačního systému. [23]

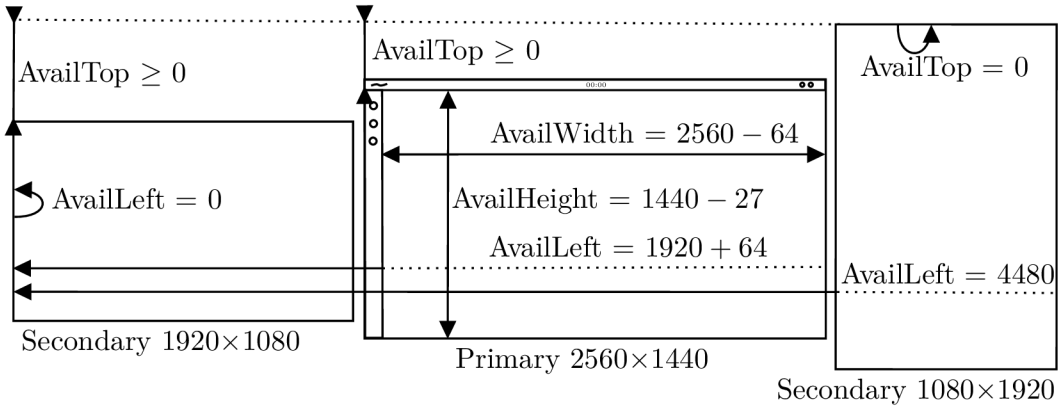
C.2 Atributy objektu `screen`

Atributy na objektu `screen` mohou být ovlivněny konfigurací v operačním systému nebo přímo v prohlížeči pomocí přiblížení stránky. Nejzajímavější je způsob výpočtu atributů `avail{Left|Top}`, který se liší mezi operačními systémy, jak je naznačeno na obrázku C.1. Z téhož obrázku lze také vyvodit následující závěry:

- Atributy `availLeft` a `availTop` mají potenciál detekovat za určitých podmínek přítomnost více zobrazovacích zařízení.
- Atributy `availWidth` a `availHeight` mají zase potenciál detekovat typ operačního systému na základě nenulového rozdílu s hodnotami atributů `width` a `height` a znalosti rozměrů prvků operačního systému.



(a) Na systému Windows se výpočty hodnot atributů `availLeft` a `availTop` odvíjí od obrazovky nastavené jako primární.



(b) Na systému Ubuntu se výpočty hodnot atributů `availLeft` a `availTop` odvíjí od nejlevěji a nejhorněji nastavené obrazovky. Nejlevější je první obrazovka, ale nejhornější může být libovolná, v tomto případě poslední.

Obrázek C.1: Na obrázku je znázorněn princip výpočtů hodnot atributů objektu `screen` na systému Windows 10 (obrázek C.1a) a Ubuntu 20.04 (obrázek C.1b). Rozložení obrazovek na obrázcích odpovídá rozložení, jak ho lze nastavit v operačním systému. Na obrázcích jsou také znázorněny typicky rozmístěné prvky operačních systému (hlavní panel a horní lišta), jejichž pozice může uživatel taktéž konfigurovat. Hodnoty jsou v jednotkách `px`. Výpočet hodnot atributů `availWidth` a `availHeight` je demonstrován pouze na primární obrazovce, ale ten samý princip výpočtu lze aplikovat i na sekundárních.

Obecně lze shrnout možnosti ovlivnění atributů na objektu `screen` v následujícím seznamu:

- **Nastavení primárního monitoru** a **rozmístění prvků v operačním systému** – `availLeft`, `availTop`, `availWidth` a `availHeight`
- **Nastavení rozložení obrazovek v operačním systému** – `availLeft` a `availTop`
- **Změna úrovně přiblížení stránky** – V rodině prohlížečů Gecko a Trident jsou hodnoty atributů `width`, `height`, `availWidth` a `availHeight` ovlivněny úrovní přiblížení stránky. První řešení problémů spočívá ve vynásobení hodnot atributů hodnotou atributu `devicePixelRatio`, což však nemusí být vždy přesné ani spolehlivé. Druhé řešení

spočívá ve výpočtu poměru mezi šířkami a výškami. Poměr hodnot se nemění s rozdílným přiblížením stránky, ale ztrácí se velká míra informace, protože více různých rozlišení dosahuje stejných poměrů např. rozlišení 1024×768 a 1280×960. [9]

C.3 Časové pásmo

Počáteční hodnota časového pásma je vždy nastavena uživatelem při instalaci operačního systému. Případná změna hodnoty časového pásma v operačním systému je ihned reflektována v prohlížeči bez nutnosti jeho restartu. V následujícím seznamu jsou uvedeny kroky nutné pro změnu časového pásma.

- **Windows** – Ovládací panely → Datum a čas → Změnit časové pásmo
- **Ubuntu** – Nastavení → Datum a čas → Časové pásmo

C.4 Jazyk hostitelského prostředí

Počáteční hodnota jazyka uživatelského rozhraní je definována jazykem instalačního souboru prohlížeče. Případná změna jazyka může být realizována v prohlížeči nebo v operačním systému a její reflektování je podmíněno restartem prohlížeče nebo operačního systému.

C.5 Instalovaná písma

Úprava písem může být způsobena buď vědomým, nebo nevědomým zásahem uživatele. Vědomá úprava písem reflektuje situaci, kdy uživatel nainstaluje/odinstaluje písmo přímo v operačním systému. Mezi nevědomé úpravy, které mohou ovlivnit detekci písem, lze zařadit následující situace:

- písma jsou dynamicky načtena webovými stránkami. Vyřazením těchto písem ze sběru se zvýší šance identifikovat uživatele napříč webovými stránkami.
- písma jsou automaticky nainstalována do operačního systému v rámci instalace nových programů. Například písmo `MT Extra` je silným indikátorem, že operační systém má nainstalovaný balík Microsoft Office. Dostupnost různých písem tedy může naznačit i přítomnost programů v operačním systému. [34, 50, 5]

C.6 Vyhlazování písma

Nastavení vyhlazování písma lze ovlivnit na úrovní operačního systému a pro aplikování změn je nutné restartovat prohlížeč.

- **Windows** – Vlastnosti systému → Výkon → Vyhladit hrany obrázkových písem. Tato konfigurace současně ovlivní i nastavení technologie ClearType.
- **Ubuntu** – Aplikace GNOME Tweaks umožňuje nastavit různé techniky vyhlazování písma, ale prohlížeče toto nastavení nereflektují.